VŠB – Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra aplikované matematiky

Využití teorie grafů pro optimalizaci BEM

Graph Theory Tools Used for BEM

2020 Bc. Tomáš Michna

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra aplikované matematiky

Zadání diplomové práce

Student: **Bc. Tomáš Michna**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 1103T031 Výpočetní matematika

Téma: Využití teorie grafů pro optimalizaci BEM
Graph Theory Tools Used for BEM

Jazyk vypracování: čeština

Zásady pro vypracování:

Při paralelizaci numerického řešení úloh pomocí BEM (Boundary element method) lze s úspěchem využít rozložení hustých matic na podmatice, jejichž řešení může probíhat na různých jádrech paralelního stroje, pokud tyto submatice odpovídají "dostatečně vzdáleným" uzlům v dané síti, která je řešena pomocí BEM. Samotné rozdělení velké husté matice na nezávislé podmatice lze popsat pomocí vhodného vrcholového barvení v grafu, který modeluje výpočetní závislost či nezávislost jednotlivých uzlů sítě.
Cílem práce je navrhnout a implementovat vhodný algoritmus, který obarví uzly dané sítě tak, aby samotná paralelizace výpočtu byla dobrá, tj. aby výpočty jednotlivých jader na sobě nezávisely.

Postup práce lze rozdělit do následujících kroků:
1) studium aktuální odborné literatury,
2) vytipování vhodných algoritmů barvení,
3) implementace algoritmů barvení a jejich zakomponování do postupu řešení pomocí BEM,
4) ověření škálovatelnosti a funkčnosti celého řešení.

Seznam doporučené odborné literatury:

- M.Garcia-Gasulla, G. Houzeaux, R. Ferrer, A. Artigues, V. López, J. Labarta, andM. Vázquez: MPI+X task-based parallelization and dynamic load ballance of finite element assembly, (on-line)
- D.B.West: Introduction to graph theory, Upper Saddle River NJ, (2001).
- K.H.Rosen: Discrete Mathematics and Its Applications, New York NY, (2007).
- odborné články a texty podle pokynů vedoucího.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **doc. Mgr. Petr Kovář, Ph.D.**

Konzultant diplomové práce: Ing. Jan Zapletal, Ph.D.

Datum zadání: 01.09.2019

Datum odevzdání: 30.04.2020

prof. RNDr. Jiří Bouchala, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 15. května 2020

Tomáš Michna

Rád bych na tomto místě poděkoval svému vedoucímu diplomové práce
doc. Mgr. Petru Kovářovi, Ph.D. a konzultantovi Ing. Janu Zapletalovi, Ph.D. za cenné rady, připomínky a čas, který mi věnovali po celou dobu vedení této diplomové práce. Hlavně pak za nadšení, které mě posouvalo v řešení této práce kupředu. V neposlední řadě děkuji své rodině a přátelům, kteří mě povzbuzovali během celého studia.

Abstrakt

Téma této práce je na rozhraní dvou matematických oblastí, kterými jsou teorie grafů a numerická matematika. V práci se zabýváme problémem kolizí, které mohou nastat během sestavení matice $\mathbb{V}$ při paralelním řešení Laplaceovy úlohy metodou hraničních prvků. Naším cílem je využít techniky teorie grafů a navrhnout efektivní algoritmus, který by rozdělil dvojice elementů do disjunktních množin. Pro všechny dvojice elementů z každé disjunktní množiny musí platit, že zápisy do paměti libovolných dvojic ze stejné množiny nesmí být v kolizi. Díky tomu mohou být výpočty na dvojicích elementů z těchto disjunktních množin počítány paralelně.

Klíčová slova: Vrcholové barvení; metoda hraničních prvků; MHP; paralelizace

Abstract

The topic of this thesis includes two mathematical areas. The first topic is graph theory and the second topic is numerical analysis. In this thesis, we focuse on the problem with collision in parallel matrix update, while solving the Laplace equation by boundary element method. The goal of this thesis is to use graph theory tools to design an effective algorithm. This algorithm has to divide pairs of elements into disjoint sets. For each pair of elements from the disjoint set the matrix update must not collide with other pair from same set. Thanks to this distribution of pairs of elements, we can compute tasks in parralell over pairs of elements from the same disjoint set.

Keywords: Vertex coloring; boundary element method; BEM; parallelism

Obsah

Seznam použitých zkratek a symbolů

BEM	–	Metoda hraničních prvků
$G = (V, E)$	–	Graf G s množinou vrcholů V a množinou hran E
$V(G)$	–	Množina vrcholů grafu G
$E(G)$	–	Množina hran grafu G
$N_G(v)$	–	Množina sousedních vrcholů vrcholu v v grafu G
$N_G[G]$	–	Uzavřená množina sousedních vrcholů vrcholu v v grafu G
$F(G), F_G$	–	Množina oblastí rovinného grafu
F	–	Počet oblastí sítě
N	–	Počet uzlů sítě
C_n	–	Cyklus na n vrcholech
K_n	–	Kompletní graf na n vrcholech
$G \square H$	–	Kartézský součin grafů G a H
$G \times H$	–	Přímý součin grafů G a H
$G \boxtimes H$	–	Silný součin grafů G a H
$\Delta(G)$	–	Největší stupeň grafu G
$\delta(G)$	–	Nejmenší stupeň grafu G
Δ	–	Laplace operátor
$c(v_i)$	–	Obarvený vrchol grafu v_i
$\chi(G)$	–	Chromatické číslo grafu G
Γ	–	Hranice oblasti
$\Omega(G)$	–	Klika grafu G
$H^{-1/2}$	–	Duální prostor k prostoru stop
$H^{1/2}$	–	Normovaný prostor stop
γ_0	–	Dirichletova stopa
γ_1	–	Neumannova stopa
$\mathcal{X}$	–	Počet časových množin
$\varphi(x)$	–	Po částech lineární funkce
$\mathcal{O}$	–	Časová / paměťová náročnost algoritmu
$\mathbb{V}$	–	Matice $\mathbb{V}$ v metodě hraničních prvků
$\mathbb{K}$	–	Matice $\mathbb{K}$ v metodě hraničních prvků
BFS	–	Algoritmus prohledávání do šířky

BEM4I	–	Paralelní matematická knihovna pro řešení parciální diferenciálních rovnic metodou BEM
LF	–	Largest first algoritmus dobrého vrcholového barvení grafu
CPU	–	Centrální procesorová jednotka
GPU	–	Grafická procesorová jednotka

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Tato diplomová práce je na pomezí dvou matematických oblastí, kterými jsou teorie grafů a numerická matematika. Úlohou, kterou se v této práci budeme zabývat, je problém vzniku kolizí, které mohou nastat při zápisu do sdílené paměti během paralelního výpočtu Laplaceovy úlohy metodou hraničních prvků. Při použití metody hraničních prvků budu využívat po částech lineární bázové funkce.

Práce je rozdělená do osmi tématických kapitol. Ve druhé kapitole si uvedeme nezbytný aparát teorie grafů, který využijeme k matematickému popisu kolizí a k návrhu vhodného řešení na základě diskrétní matematiky. Ve třetí kapitole shrneme základy teorie řešení parciálních diferenciálních rovnic metodou hraničních prvků. Ve čtvrté kapitole pomocí pojmů teorie grafů popíšeme kolize, které vznikají při paralelním sestavení matice $\mathbb{V}$ během řešení Laplaceovy úlohy metodou hraničních prvků. V páté kapitole se zaměříme na efektivní návrh řešení problému s kolizemi. Našim cílem je vymyslet efektivní algoritmus, který má za úkol rozdělit dvojice elementů, nad kterými interagují bázové funkce, do disjunktních množin tak, že jakékoliv dvě dvojice elementů ze stejné množiny nezapisují svůj výsledek do společné buňky v matici $\mathbb{V}$. Díky této podmínce mohou být výpočty pro dvojice ze stejné množiny vykonávány paralelně a bez vzniku kolizí, které bychom jinak museli ošetřit atomickou sekcí. V šesté kapitole se zaměříme na vylepšení našeho řešení. V sedmé kapitole se budeme zabývat numerickými experimenty, které budeme testovat na superpočítači Barbora [6]. Jedním z cílů práce je implementace našeho řešení do paralelní knihovny BEM4I [11]. V této kapitole srovnáme naše řešení sestavení matice $\mathbb{V}$ s řešením pomocí atomické sekce, které je již implementováno v knihovně BEM4I [11]. Dále se v této části zaměříme na čas sestavení výše zmíněných disjunktních množin dvojic elementů. V poslední části zhodnotíme dosažení cílů této práce.

2 Teorie grafů

Pojem grafu zavedl Leonhard Euler roku 1736, když řešil úlohu sedmi mostů ve městě Královci v dnešním Kaliningradu. Od té doby se začalo vyvíjet odvětví diskrétní matematiky zvané teorie grafů. S výsledky z teorie grafů se setkáváme dnes a denně. Jeden z příkladů je úloha nalezení nejkratší trasy v silniční mapě pomocí GPS, nebo nalezení nejkratší cesty k dané webové stránce napříč internetem. Síla teorie grafů tkví v abstraktním popisu problémů a odhlédnutí od zbytečných detailů. Tuto kapitolu si rozdělíme na tři hlavní podkapitoly. V první části popíšeme základní poznatky a pojmy z teorie grafů, které budeme v této práci využívat. V druhé kapitole se zaměříme na dobré vrcholové barvení grafu. Pomocí dobrého vrcholového barvení umíme rozdělit množinu vrcholů do disjunktních skupin, pokud víme, jaké vrcholy spolu nesmí být ve společné skupině. V poslední třetí části rozebereme algoritmy dobrého vrcholového barvení.

2.1 Úvod do teorie grafů

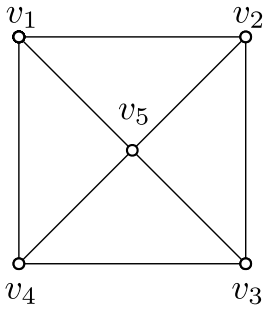
V této kapitole popíšeme základní pojmy týkající se teorie grafů. Zmíníme, pojem grafu, dále se zaměříme na třídy grafů. Nejprve zaveďme samotný pojem grafu.

Definice 1 *Jednoduchý neorientovaný (obyčejný) graf G je uspořádaná dvojice (V, E), kde V je neprázdná množina vrcholů a E je množina dvouprvkových podmnožin V. Prvkům v E říkáme hrany.*

V této práci budeme pracovat pouze s jednoduchým typem grafů. Graf budeme značit velkými tiskacími písmeny G nebo H. Malými písmeny v, v_i, u budeme značit vrcholy daného grafu. Množinu $V(G)$ nazveme množinou vrcholů grafu G a množinu $E(G)$ nazveme hranovou množinou grafu G.

Příklad 1
Uveďme nyní příklad grafu G, který je znázorněn na Obrázku 1. Vrcholová množina grafu G je rovna $V(G) = \{v_1, v_2, v_3, v_4, v_5\}$. Hranová množina grafu G je
$E(G) = \{\{v_1, v_2\}, \{v_1, v_5\}, \{v_2, v_3\}, \{v_2, v_5\}, \{v_3, v_4\}, \{v_3, v_5\}, \{v_4, v_1\}, \{v_4, v_5\}\}$. Zaveďme si úmluvu, že hranu $e = \{u, v\}$ budeme v textu nadále značit zkráceným zápisem $e = uv$. Řádem grafu G budeme rozumět velikost množiny $V(G)$ kterou budeme značit $|V(G)| = n$. Počet hran grafu G budeme značit takto $|E(G)|$.

Pod pojmem *jednoduchý graf* budeme v této práci rozumět takový graf G, ve kterém se nevykytuje žádná hrana $e = v_i v_i$, která se nazývá smyčka. Taktéž se v jednoduchém grafu nemůže



Obrázek 1: Jednoduchý graf G

vyskytnout hrana v_iv_j více krát, což by znamenalo, že graf G obsahuje multihrany. Nyní zavedeme pojmy týkající se množiny sousedních vrcholů a stupně daného vrcholů. Tyto pojmy v další podkapitole využijeme při odhadech počtů barev a v Brooksově větě.

Definice 2 *[7] Mějme jednoduchý graf G, potom dva vrcholy $u, v \in V(G)$ nazvěme závislé (sousední), pokud v hranové množině $E(G)$ existuje hrana uv, v opačném případě nazvěme vrcholy u a v nezávislé. Množinu všech sousedních vrcholů k vrcholu u v grafu G budeme značit $N_G(u)$.*

Příklad 2
Pro objasnění pojmu množiny sousedních vrcholů využijeme graf G z Obrázku 1. Pro vrchol v_5 dostaneme množinu sousedních vrcholů $N_G(v_5) = \{v_1, v_2, v_3, v_4\}$. Pro vrchol v_1 dostáváme množinu sousedních vrcholů $N_G(v_1) = \{v_2, v_4, v_5\}$.

Definice 3 *[7] Stupeň vrcholů v je počet hran, se kterými je daný vrchol v incidentní, a značí se* $\deg(v)$.

Poznámka 1 Vztah mezi stupněm vrcholů a množinou sousedních vrcholů se dá vyjádřit $\deg(v) = |N(v)|$. Tento vztah platí pro libovolný vrchol v každém jednoduchém grafu. Díky zavedené Definici 3 stupně vrcholu, můžeme definovat maximální (Δ) a minimální (δ) stupeň vrcholů grafu G.

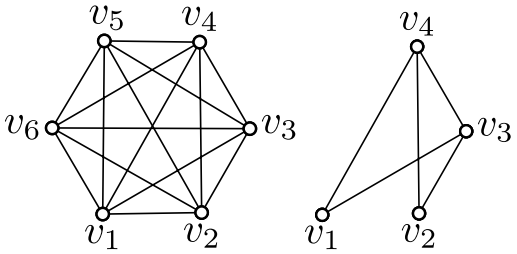
$$\begin{aligned} \Delta = \Delta(G) &:= \max_{v \in V(G)} \{\deg(v)\} \\ \delta = \delta(G) &:= \min_{v \in V(G)} \{\deg(v)\} \end{aligned} \tag{1}$$

Příklad 3
V tomto příkladu, si opět vezmeme graf G z Obrázku 1. Pro vrchol v_5 dostáváme stupeň vrcholů $\deg(v_5) = 4$. Pro ostatní vrcholy v_i v grafu G , kde $i \in \{1, 2, 3, 4\}$ dostáváme $\deg(v_i) = 3$. Není těžké určit $\Delta(G) = 4$ a $\delta(G) = 3$.

Definice 4 *[7] Mějme dán graf $G = (V, E)$. Řekneme, že graf $H = (V', E')$ je podgrafem grafu G, jestliže $V' \subset V$ a současně $E' \subset E$.*

Příklad 4

Pro objasnění pojmu podgraf grafu uvádíme na Obrázku 2 příklad grafu $G = (V, E)$ a jeho podgraf $H = (V', E')$, kde vrcholová množina $\mathbb{V}'$ je tvořena vrcholy v_1, v_2, v_3, v_4 a množina hran E' je tvořena hranami $v_1v_3, v_1v_4, v_2v_3, v_2v_4, v_3v_4$.



(a) graf G (b) graf H

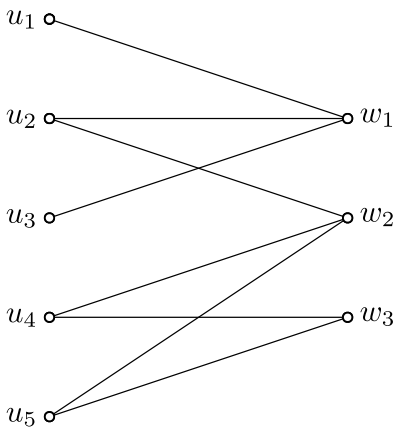
Obrázek 2: Příklad podgrafu grafu G

V další části této podkapitoly zmíníme speciální třídy grafů, které budou bipartitní grafy kompletní grafy a rovinné grafy. Bipartitní graf využijeme při modelování problému při přístupu do společné paměti ve výpočtu řešení Laplaceovy úlohy v metodě hraničních prvků. Kompletní graf nám poslouží k uvedení klikového čísla grafu díky, kterému budeme lépe schopni určit dolní odhad počtů barev při dobrém vrcholovém barvení.

Definice 5 *[15] graf G nazveme bipartitním grafem, jestliže vrcholová množina $V(G)$ je dána sjednocením dvou disjunktních (neprázdných) množin vrcholů. Jednotlivé vrcholové množiny se nazývají partity.*

Příklad 5

Uvedeme zde příklad bipartitního grafu G, který bude tvořen dvěma partitami vrcholů $U = \{u_1, u_2, u_3, u_4, u_5\}$ a $W = \{w_1, w_2, w_3\}$, viz Obrázek 3.



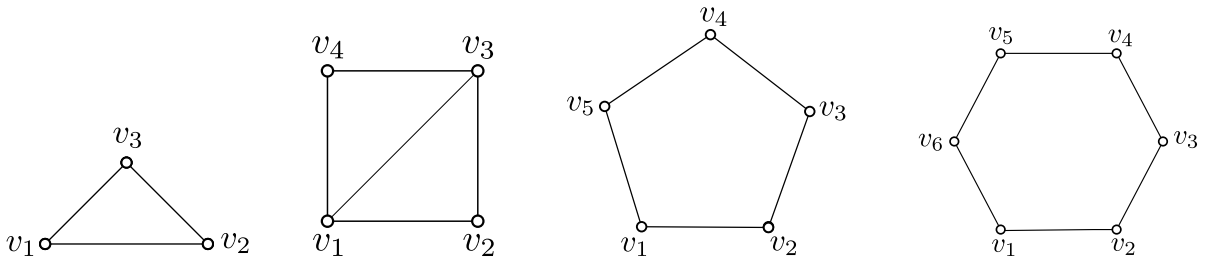
Obrázek 3: Bipartitní graf G

S bipartitními grafy se v aplikacích mimo jiné modelují vztahy mezi dvěma různými množinami. Prvky daných množin se převedou na vrcholy dané partity a hrany mezi vrcholy v jednotlivých partitách modelují daný vztah. Například jednu partitu mohou tvořit jednotlivé předměty na vysoké škole a druhou množinou mohou být studenti. Hrana mezi předmětem a studentem může mít význam zápisu daného předmětu studentem v daném akademickém roce.

Další třídy grafů, které zmíníme budou cykly a kompletní grafy.

Definice 6 *[7] Mějme jednoduchý graf $G = (V, E)$. Nazvěme tento graf cyklem C_n, jestliže $V = \{v_1, v_2, \ldots, v_n\}$, $|V(G)| \geq 3$ a současně platí, že $E(G) = \{v_1v_2, v_2v_3, \ldots, v_{n-1}v_n, v_nv_1\}$.*

Příklad 6
Uvedeme zde příklad pár cyklů, viz Obrázek 4. Dolním symbolem n v označení C_n, je určena velikost vrcholové množiny cyklu, tj. $|V(C_n)| = n$.

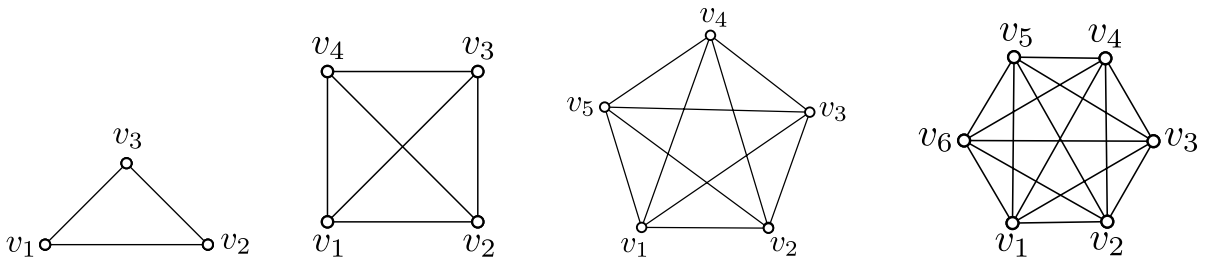


(a) graf C_3 (b) graf C_4 (c) graf C_5 (d) graf C_6

Obrázek 4: Příklady cyklů

Definice 7 *[7] Graf s označením K_n nazveme kompletním grafem, pokud v tomto grafu jsou každé dva vrcholy sousední.*

Příklad 7
Uvedeme zde příklad pár kompletních grafů, viz Obrázek 5. Dolním symbolem n v označení K_n, je určena velikost vrcholové množiny kompletního grafu, tj. $|V(K_n)| = n$.



(a) graf K_3 (b) graf K_4 (c) graf K_5 (d) graf K_6

Obrázek 5: Příklady kompletních grafů

Díky zavedeným pojmům podgraf grafu a kompletní graf můžeme zavést klikové číslo grafu G.

Definice 8 *[7] Klikové číslo grafu G, značíme $\omega(G)$. Jedná se o počet největšího kompletního grafu (kliky), který je podgrafem G.*

2.2 Planární grafy

V této kapitole se zaměříme na popis planárních grafů. Tato oblast teorie grafů nachází uplatnění například v elektrotechnice při návrhu tištěných spojů. Klasickou úlohou z této oblasti může být rozhodnutí, zda je možné schéma elektrotechnického popisu přístroje přenést na jednu vrstvu tištěného spoje, nebo zda je potřeba použit více vrstev tištěných desek. Tuto kapitolu jsme zde zařadili proto, že některé sítě 3D těles se dají reprezentovat pomocí planárního grafu. Proto zde uvedeme některé vlastnosti tykající se planárních grafů. Než se k samotným výsledkům dostaneme, je potřeba uvést základní definici planárního a rovinného grafu.

Definice 9 *[7] Graf je rovinný, jestliže máme dáno jeho nakreslení do roviny takové, že žádné dvě křivky, které odpovídají hranám, nemají společné body kromě koncových bodů (vrcholů). Dále graf je planární, jestliže existuje jeho rovinné nakreslení, tj. jestliže je možné jej nakreslit do roviny tak, aby žádné dvě hrany neměly společné body, než koncové).*

Nyní, když jsem zavedli základní definici rovinného a planárního grafu, uvedeme zde známou Eulerovou větu pomocí, které budeme schopni odhadnout některé vlastnosti rovinných grafů.

Věta 1 *[7]* ***Eulerův vzorec***
Pro libovolný souvislý rovinná graf G platí $|V(G)| - |E(G)| + |F(G)| = 2$.

Poznámka 2 Označením $F(G)$ ve Větě 1 budeme rozumět množinu oblastí rovinného grafu G. Pojem oblast má smysl zavést pouze pro rovinné grafy.

Důkaz Eulerova vzorce je možné nalézt v [7]. Uveďme si jeden z důsledků Eulerova vzorce. Tento důsledek nám dá odpověď na maximální počet oblastí rovinného grafu vzhledem k počtu vrcholů.

Důsledek 1 *Počet oblastí souvislého planárního grafu G složeného pouze z trojúhelníku je $|F(G)| = 2|V(G)| - 4$.*

Důkaz Mějme dán souvislý graf G, kde každá oblast grafu G je složená z trojúhelníků. To znamená, že každou oblast tvoří tři hrany. Víme, že každá hrana je na hranicí právě dvou oblastí. Nyní sečteme počet všech hran v grafu G. Za každou oblast obdržíme tři hrany a počet oblastí grafu G je $|F(G)|$, tj. $3|F(G)|$. Protože, jsme každou hranu započítali dvakrát dostáváme

rovnici $3|F(G)| = 2|E(G)|$. Nyní si vyjádříme z předchozí rovnice počet hran, které následně dosadíme do Eulerova vzorce 1, tj. $|E(G)| = \frac{3}{2}|F(G)|$.

$$\begin{aligned} |V(G)| - |E(G)| + |F(G)| &= 2, \\ |V(G)| - \frac{3}{2}|F(G)| + |F(G)| &= 2, \\ |V(G)| - \frac{1}{2}|F(G)| &= 2, \\ |V(G)| - 2 &= \frac{1}{2}|F(G)|, \\ |F(G)| &= 2|V(G)| - 4. \end{aligned}$$

■

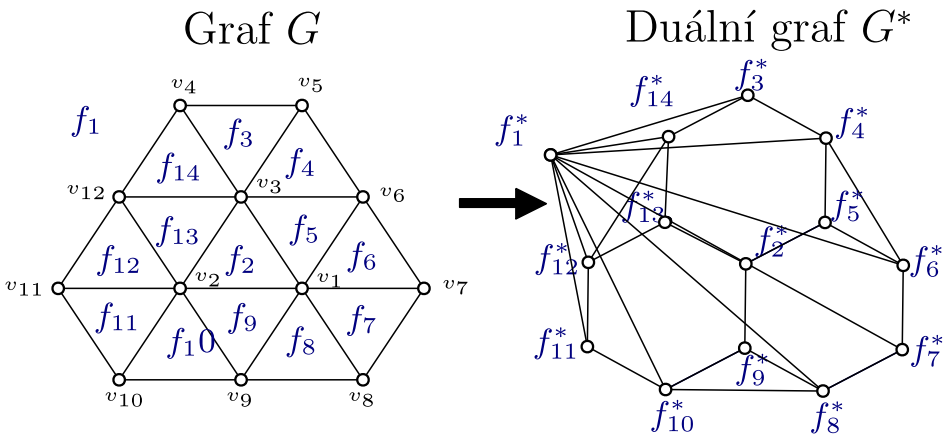
Zmíníme ještě jeden důsledek, který vyplývá z Eulerova vzorce 1.

Důsledek 2 *[7] Pro každý planární graf G platí $\delta(G) \leq 5$.*

Důkaz tohoto důsledku lze nalézt v [7]. Dále zde definujeme pojem duálního grafu, se kterým budeme pracovat později.

Definice 10 *[7]* ***Duální graf***
Mějme rovinný graf G, s množinou hran $e_1, e_2, \ldots, e_p$ a množinou oblastí $f_1, f_2, \ldots, f_q$. Duální graf G^* grafu G s množinou $V(G^*) = \{f_1^*, f_2^*, \ldots, f_q^*\}$ a množinou hran $E(G^*) = \{e_1^*, e_2^*, \ldots, e_p^*\}$, kde dva vrcholy f_i^*, f_j^* budou spojené hranou e_k^* jestliže hrana e^k je hranicí oblastí f_i^* a f_j^*.

Na Obrázku 6 uvádíme příklad rovinného grafu G a jeho duálu G^*. Dá se dokázat věta, že každý duální graf je současně i planární grafem a tedy k němu existuje rovinné nakreslení, ve kterém se žádne dvě hrany nekříží.



Obrázek 6: Graf G a jeho duální graf G^*

2.3 Vrcholové barvení grafu

V této kapitole zmíníme základní pojmy dobrého vrcholového barvení grafu, zavedeme chromatické číslo grafu, uvedeme horní a dolní odhady chromatického čísla a zmíníme také Brooksovou větu.

Dříve než tak učiníme, uveďme na začátek jednu motivační aplikaci vrcholového barvení. Bude se jednat o tzv. skladovací problém. Představme si, že jsme imaginárním vlastníkem restauračního zařízení a vlastníme sklad potravin. Ve skladu máme několik potravin např. maso, vejce, zeleninu, ovoce. Z hygienických předpisů víme, že některé potraviny nesmí být uchovávaný společně. Jelikož chceme splnit hygienické předpisy, bude nás zajímat, kolik oddělených míst musíme ve skladu takto vytvořit. Vrcholy v našem grafu budou reprezentovat jednotlivé potraviny v našem skladu. Hranami v grafu budeme modelovat vztah, že dané potraviny nesmí být umístěny ve stejném skladovém prostoru. Pokud takhle vzniklý graf dobře vrcholově obarvíme, dostaneme počet oddělených míst ve skladu, které bude potřeba vytvořit. Jednotlivé potraviny stejné barvy, budou moci být umístěny ve stejném skladovacím místě.

Definice 11 *[7] Vrcholové barvení grafu G je zobrazení c vrcholové množiny $V(G)$ do množiny barev B. Obvykle je $B = \{1, 2, \ldots, k\}$. Je-li $|B| = k$, budeme zobrazení c nazývat vrcholové k-barvení (nebo jen k-barvení) grafu G. Řekneme, že vrchol v je obarven barvou i jestliže $c(v) = i$. Číslu i říkáme barva vrcholu. Vrcholové barvení se nazývá dobré, jestliže žádné dva sousední vrcholy nejsou obarveny stejnou barvou.*

V Definici 11 jsme zavedli dobré vrcholové barvení, které využijeme při řešení kolizí během interakcí uspořádaných dvojic oblastí, které přistupují do globální matice v metodě hraničních prvků. Se zavedeným pojmem vrcholové barvení grafu G, můžeme zmínit termín barevné třídy.

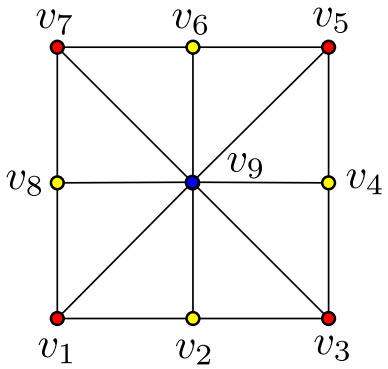
Definice 12 *[9] Barevnou třídou grafu G, nazveme všechny vrcholy grafu G, které mají přiřazenou příslušnou barvu v dobře obarveném vrcholovém grafu G. Pro každou příslušnou barvu $i \in \{1, 2, \ldots, k\}$, definujeme barevnou třídu jako $\{v \in V : c(v) = i\}$*

Na barevné třídy se můžeme dívat jako na disjunktní rozklad množiny vrcholů V grafu G. Nyní zavedeme chromatické číslo grafu $\chi(G)$, kterým budeme značit nejmenším počtem barev na dobré vrcholové obarvení grafu G.

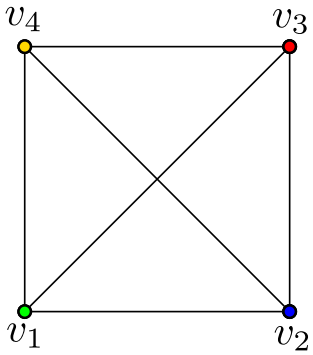
Definice 13 *[7] Řekněme, že graf G je vrcholově, k-obarvitelný (nebo jen k-obarvitelný), jestliže existuje jeho dobré vrcholové barvení k barvami. Nejmenší číslo k takové, že graf G je vrcholově k-obarvitelný, se nazývá chromatické číslo grafu G a značí se $\chi(G)$. graf G se nazývá vrcholově k-chromatický, jsou-li jeho vrcholy dobře obarveny k barvami a platí $k = \chi(G)$.*

Příklad 8

Nyní uveďme příklady grafů, které jsou dobře vrcholově obarvené, viz Obrázek 7. Pro první graf G z Obrázku 7(a) jsme použili tři barvy a pro graf K_4 z Obrázku 7(b) jsme použili čtyři barvy. V následujících podkapitolách uvedeme horní a dolní odhady chromatického čísla pro různé grafy. Dále pak zjistíme, zda dané grafy z Obrázku 7 jsou k-chromatické.



(a) Dobře obarvený graf G



(b) Dobře obarvený graf K_4

Obrázek 7: Příklady dobře vrcholově obarvených grafů

Věta 2 *[7] Je-li $G \cong K_n$, nebo $G \cong C_{2n+1}$, potom $\chi(G) = \Delta(G) + 1$.*

Důkaz Věty 2 lze nalézt v [7]. Díky této Větě 2 víme, že graf K_4 je vrcholově 4-chromatický, neboť $\Delta(K_4) = 3$. Podle Věty 2 je chromatické číslo $\chi(K_4) = \Delta(K_4) + 1 = 3 + 1 = 4$, což je i případ Obrázku 7(b). V následujícím odstavci se zaměříme na dolní odhad chromatického čísla v libovolném grafu G a zjistíme, zda graf z Obrázku 7(a) je vrcholově 3-chromatický.

Lemma 1 *[7] Pro libovolný graf G platí $\chi(G) \geq \omega(G)$.*

Ačkoliv důkaz Lemmatu 1 je v [7], ponechán jako cvičení, je zřejmé, že na dobré vrcholové pokrytí podgrafu, který je největší klikou daného grafu G, bude potřeba minimálně $\omega(G)$ barev, viz Věta 2. Přidáváním dalších vrcholů a hran se chromatické číslo grafu nemůže snížit. Proto se jedná o dolní odhad chromatického čísla. Na Obrázku 7(a) vidíme, že největší klika, která je podgrafem grafu G, je K_3, např. vrcholy (v_7, v_6, v_9). Klikové číslo grafu G, tj. $\omega(G) = 3$. Protože se nám podařilo nalézt dobré obarvení grafu G pomocí tří barev a podle Lemmatu 8 menší počet barev už nejde použít. Graf G je vrcholově 3-chromatický. Dále se podíváme na horní odhad chromatického čísla grafů.

Lemma 2 *[7] Pro libovolný graf G platí $\chi(G) \leq \Delta(G) + 1$*

Lemma 2 využijeme v pozdější fázi v algoritmech jako horní odhad minimálního možného počtu barev. Nyní zmíníme Brooksovou větu, která nám sníží horní odhad počtu barev pro grafy, které nejsou kompletní grafy a mají nejvyšší stupeň grafu alespoň 3.

Věta 3 *[7]* ***Brooksova***

Mějme souvislý graf G s $\Delta(G) \geq 3$, který je různý od kompletního grafu. Potom platí $\chi(G) \leq \Delta(G)$.

Důkaz věty lze nalézt v [7].

2.4 Algoritmy dobrého vrcholového barvení

Dříve, než zmíníme samotné algoritmy, uveďme pojem asymptotické složitosti, díky které budeme moct posuzovat časovou a paměťovou náročnost algoritmů.

Definice 14 *[10] Nechť $f, g : \mathbb{N} \to \mathbb{R}$. Řekněme, že funkce $f(n)$ je třídy $\mathcal{O}(g(n))$, jestliže*

$$(\exists c > 0)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0) : f(n) \leq c \cdot g(n).$$

Funkci $g(n)$ se pak říká asymptotický horní odhad funkce $f(n)$.

V teoretické informatice je známá úloha rozhodnutí, zda zadaný graf je bipartitní či nikoliv. Tato úloha s rozhodnutím bipartitního grafu se dá převést na úlohu, zda je graf 2-chromatický tak, že jednu partitu obarvíme barvou 1 a druhou partitu barvou 2.

K samotnému určení, zda daný souvislý graf je bipartitní, nám postačí provést Algoritmus 1. Jedná se o modifikovaný algoritmus prohledávání do šířky (BFS). Modifikace se týká přidáním testování úrovně následovníků na 18. až 20. řádku. Jestli nenastala situace s cyklem liché délky, který vlastnost bipartitního grafu poruší, Algoritmus 1 vrátí *False* na 19. řádku. Pokud pro zadaný graf G dostaneme výsledek od Algoritmu 1 *True*, znamená to, že graf G je bipartitní a lze dobře obarvit dvěma barvami. Samotné určení barev v dobrém vrcholovém barvení určíme z pole D, kde vrcholům sudé úrovně přiřadíme barvu i a vrcholům liché úrovně barvu j. Bohužel algoritmicky rozhodnout, zda graf G je k-chromatický, pro $k \geq 3$ je NP-úplná úloha [10]. Laicky řečeno, není dosud znám algoritmus s polynomiální časovou náročností, který by uměl rozhodnout, zda daný graf je k-chromatický, kde $k \geq 3$.

Algorithm 1 [10] Testování bipartitity souvislého grafu pomocí algoritmu prohledávání do šířky

Require: souvislý graf $G = (V, E)$, $E \neq \emptyset$ a počáteční vrchol $v_0 \in V$

1: **for all** vrcholy $v \in V$ **do**
2: $stav(v) \leftarrow nenalezený$
3: $D(v) = \emptyset$
4: $P(v) \leftarrow \emptyset$
5: **end for**
6: $stav(v_0) \leftarrow otevřený$
7: $D(v_0) \leftarrow 0$
8: Založíme frontu Q a vložíme do ní vrchol v_0
9: **while** Dokud je fronta Q neprázdná **do**
10: Odebereme první vrchol z Q a označíme ho v
11: **for all** Pro všechny následovníky w vrcholu v **do**
12: **if** Je-li $stav(v) = nenalezený$ **then**
13: $stav(v) \leftarrow otevřený$
14: $D(w) = D(v) + 1$
15: $P(w) = v$
16: přidáme w do fronty Q
17: **end if**
18: **if** $D(v) == D(w)$ **then**
19: **return** False
20: **end if**
21: **end for**
22: $stav(v) \leftarrow uzavřený$
23: **end while**
24: **return** True

Jedním z algoritmů, který umí rozhodnout a při úspěšném rozhodnutí i dobře vrcholově obarvit graf G pomocí k barev, je Algoritmus 2. Tento Algoritmus 2 je založený v nejhorším případě na vyzkoušení všech možných kombinací obarvení grafu G tak, aby bylo dobré. Tento postup se nazývá technika hrubé síly, anglicky *brute force*. Tento Algoritmus 2 využívá techniku zpětného vracení ve stromě zpracovaných vrcholů a opravy barvy zvané *backtracking*. Algoritmus 2 se v praktických aplikacích nepoužívá, protože časová náročnost Algoritmu 2 je exponenciální, tj. $\mathcal{O}(k^n)$, kde k je počet barev, které slouží k dobrému obarvení grafu G a n počet vrcholů grafu G.

2.4.1 Popis *brute force* Algoritmu 2 k nalezení dobrého vrcholového obarvení grafu G pomocí k barev

V Algoritmu 2 reprezentujeme graf G pomocí matice sousednosti uložené v proměnné *AdjancencyMatrix*. Počet vrcholů grafu G v Algoritmu 2 značíme *NodeCount* a počet barev, které bude Algoritmus 2 zkoušet obarvit graf G, je označeno celočíselnou proměnnou k. Na začátku Algoritmu 2 je potřeba vytvořit pole *ColorArray*, které v sobě bude uchovávat barvy obarvených vrcholů grafu G. Dále v Algoritmu 2 se nachází dvě funkce *GraphColor* a *isValid*. Funkce

GraphColor slouží k obarvení celého grafu G tím, že zkouší různé kombinace obarvení vrcholů. V případě úspěšného obarvení grafu G pomocí k barev na 10. řádku Algoritmu 2 vytiskne pole *ColorArray* a vrátí hodnotu *True* jako návratovou hodnotu úspěchu. V případě neúspěšného obarvení je vrácena hodnota *False* na 15. řádku Algoritmu 2. Zmiňme, že na 8. řádku Algoritmu 2 je rekurzivní zavolání samotné funkce *GraphColor*. Co se týče funkce *isValid*, tak ta zkontroluje sousední vrcholy vrcholu i a vrátí odpověď, zda je možné vrchol i obarvit barvou c tak, aby nebyla porušena podmínka dobrého vrcholového barvení grafu G. Na 27. řádku Algoritmu 2 je zavolána funkce *GraphColor* s prvním vrcholem grafu G označeným číslem 0.

Algorithm 2 Nalezení dobrého vrcholového barvení grafu G pomocí k barev metodou *backtracking*

```
Require: AdjancencyMatrix, NodeCount, k
 1: ColorArray ← ZeroArray[NodeCount];
 2: function GraphColor(Node i)
 3:     ColorResult = False
 4:     for color c ∈ {1, 2, ..., k} do
 5:         if isValid(i, c) then
 6:             ColorArray[i] ← c
 7:             if i < NodeCount then
 8:                 ColorResult ← ColroResult OR GraphColor(i+1)
 9:             else
10:                 Print(ColorArray, NodeCount)
11:                 ColorResult ← True
12:             end if
13:             ColorArray[i] ← 0
14:             if ColorResult == True then
15:                 return True
16:             end if
17:         end if
18:     end for
19:     return False
20: end function
21:
22: function isValid(Node i, color c)
23:     for j ∈ {0, 1, ..., NodeCount − 1} do
24:         if AdjancencyMatrix[i][j] == 1 AND ColorArray[j] == c then
25:             return False
26:         end if
27:     end for
28:     return True
29: end function
30:
31: result = GraphColor(0)
```

Poznámka 3 Pokud bychom v Algoritmu 2 zkoušeli iterativně zvedat hodnotu k a obarvovali graf G do prvního úspěšného obarvení. Za předpokladu neomezeného výpočetního času a výpočetního výkonu, obdrželi bychom obarvený graf G pomocí $\chi(G)$ barev. Bohužel už pro malý počet vrcholů $n = 300$ a $k = 6$, dostáváme v nejhorším případě cca. $2.78 \cdot 10^{233}$ možností, které by musel Algoritmus 2 otestovat. To je více než, je počet atomů v celém vesmíru (odhaduje se 10^{100}).

2.4.2 Heuristické algoritmy k nalezení dobrého vrcholového

Protože předchozí *brute force* Algoritmus 2 sice vedl k obarvení grafu G pomocí $\chi(G)$ barev, ale za cenu velké časové náročnosti. Proto se v praxi používají heuristické algoritmy, které nezaručují obarvení grafu G pomocí optimálního počtu $\chi(G)$ barev v dobrém vrcholovém barvení grafu G, ale snaží se k tomuto optimu přiblížit. Jejich výhoda je mešní časová složitost oproti Algoritmu 2 založeném na *brute force* technice. Tyto heuristické algoritmy si nyní představíme. V praxi nacházejí využití, díky své rychlosti, která je způsobená polynomiální časovou náročnosti.

Algorithm 3 [8] Hladový algoritmus k nalezení dobrého vrcholového barvení grafu G

```
1: function GreedyColor(Graph G, Sequence K)
2:     for v_i ∈ K do
3:         přiřaď vrcholu v_i nejnižší hodnotu barvy na svém okolí
4:     end for
5: end function
```

Ačkoliv v Algoritmu 3 z [8] není zmíněna struktura, do které se ukládá barva, my však budeme uvažovat, že jim bude celočíselné jednorozměrné pole *ColorsArray*. Pole *ColorsArray* bude mít velikost $n = |N_G|$, kde $|N_G|$ je počet vrcholů grafu G. Tento Algoritmus 3 má časovou složitost $\mathcal{O}(n + m)$, kde $m = |E_G|$ je velikost množiny hran. Časová složitost je způsobená tím, že Algoritmus 3 musí přistoupit ke každému vrcholu grafu G a přiřadit mu barvu. Navíc musí zkontrolovat zda nově přiřazená barva již není použitá na okolí vrcholu. Prostorová složitost Algoritmu 3 je $\mathcal{O}(n)$, protože je potřeba si ke každému vrcholu uchovávat informaci o přiřazené barvě v poli *ColorsArray*.

Poznámka 4 Tento algoritmus využijeme hlavně v implementační částí této práce, kvůli své malé časové náročnosti, když budeme dobře vrcholově barvit grafy.

Dále můžeme zmínit rozšíření tohoto hladového Algoritmu 3. Jedním z rozšíření je *LF* (*largerst-first*) algoritmus založený na metodě obarvení vrcholů největšího stupně grafu G.

Algorithm 4 [8] LF Algoritmus k nalezení dobrého vrcholového barvení grafu G

```
1: function LF-Color(Graph G, Sequence K)
2:     K ← Sestupně uspořádejme vrcholy grafu G do posloupnosti podle stupňů vrcholů
3:     GreedyColor(G, K)
4: end function
```

Autoři tohoto Algoritmu 4 v [8] zmiňují, že je možné tento Algoritmus 4 *LF* (*largerst-first*) implementovat s časovou složitostí $\mathcal{O}(n)$. Třídění vrcholů v tomto Algoritmu 4 je jednou z nejvíce časově náročnou operací. Nejspíše autoři předpokládají využít modifikovaný *Bucket-sort* algoritmus, který má časovou náročnost $\mathcal{O}(n)$. Tuto modifikaci si představíme. Jedna z myšlenek může vypadat tak, že si vytvoříme pomocné celočíselné pole *degree* o velikosti n. Následně během lineárního průchodu grafu G si uložíme pro každý vrchol grafu G informaci o stupni vrcholů do pole *degree*. V další fázi třídícího algoritmu je třeba zjistit minimální ($\delta(G)$) a maximální ($\Delta(G)$) stupeň v grafu G. Tento krok, kdy zjišťujeme $\delta(G)$ a $\Delta(G)$ se dá spojit s vyplněním pole *degree*. Ve chvíli, kdy máme zjištěné $\delta(G)$ a $\Delta(G)$ a naplněné pole *degree*, můžeme přistoupit k tvorbě jednodimenzionálního pole *bucket*. Pole *bucket* bude v sobě obsahovat odkazy na spojové seznamy indexů vrcholů. Velikost pole *bucket* bude $\Delta(G) - \delta(G) + 1$. Nyní již stačí projít pole *degree* a správně zařadit indexy vrcholů do pole *bucket*. Samotné zařazení vrcholu i do pole *bucket* bude probíhat tak, že do *bucket*$[\Delta(G) - $*degree*$[i]]$ přiřaď nový index i. Nyní se v poli *bucket* nachází rozdělené vrcholy grafu G podle stupňů vrcholů. Posledním krokem je vytvoření sekvence K. Sekvenci K budeme reprezentovat polem o velikosti n. Nyní již stačí projít jednotlivé spojové seznamy uložené v poli *bucket*. Ve stejném pořadí v jakém jsme navštívili index vrcholu i v poli *bucket*, tak pod takovým pořadím se bude v sekvenci K nacházet index vrcholu i. Nyní již jsou v sekvenci K uspořádané sestupně indexy vrcholů podle stupně vrcholů.

Další hladový algoritmus, který je zmíněn v [8] je Algoritmus 5. Oproti Algoritmu 3 využívá přebarvení vrcholů ve chvíli, kdy má použit novou barvu. V článku [8] je uvedená časová složitost algoritmu $\mathcal{O}(n + m)$. Algoritmus 5 nemusí vždy správně obarvit vrcholy, tak aby počet barev byl rovný $\chi(G)$, jedná se proto o heuristický algoritmus.

Algorithm 5 [8] algoritmus s výměnou barvy k nalezení dobrého vrcholového barvení grafu G

```
1: function Color-with-Interchange(Graph G, Sequence K)
2:     for v_i ∈ K do
3:         if pokud vrchol v_i vyžaduje použit novou barvu then
4:             Pokusme se přebarvit vrcholy na okolí vrcholu v_i na jinou barvu
5:         end if
6:         přiřaď vrcholu v_i nejnižší hodnotu barvy na svém okolí
7:     end for
8: end function
```

2.5 Operace s grafy

V této malé podkapitole uvedeme pár základních operací s grafy. Důležitým termínem, v této sekci je popis silného součinu grafů. Tuto operaci bychom mohli pojmenovat silný součin, bohužel v české literatuře tato operace nemá přesný termín. Při operaci s grafy, budeme předpokládat, že se bude jednat o jednoduché grafy. Proto abychom mohli zavést operaci silného součinu grafů je nezbytné uvést definici kartézského součinu grafů a přímého součinu grafů.

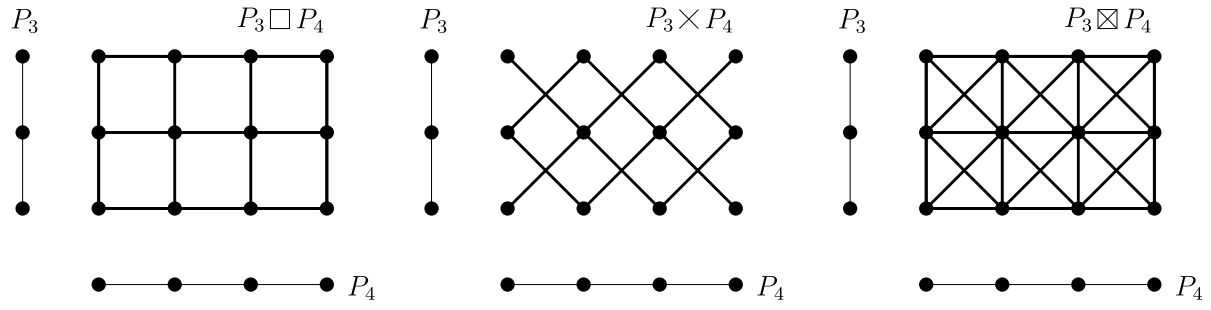
Definice 15 *[5]*
Kartézský součin grafů G a H nazveme graf $G\square H$ takový, že vrcholová množina grafu $G\square H$ je $V(G\square H) = V(G) \times V(H)$. Dva vrcholy (g, h) a (g', h') jsou spojené hranou v grafu $G\square H$, právě když platí $g = g'$ a $hh' \in E(H)$ nebo $h = h'$ a $gg' \in E(G)$.

Definice 16 *[5]*
Přímý součin grafů G a H nazveme graf $G \times H$ takový, že vrcholová množina grafu $G \times H$ je $V(G \times H) = V(G) \times V(H)$. Dva vrcholy (g, h) a (g', h') jsou spojené hranou v grafu $G \times H$, právě když platí $gg' \in E(G)$ a $hh' \in E(H)$.

Nyní již můžeme zavést Silný součin jak jsme zmínili v úvodu této podkapitoly. Symbolem $\times$ ve výrazu $V(G) \times V(H)$ rozumíme tak, že se jedná o kartézský součin prvků z množin.

Definice 17 *[5] Silný součin grafů G a H nazveme graf $G \boxtimes H$ takový, že vrcholová množina grafu $G \boxtimes H$ je $V(G \boxtimes H) = V(G) \times V(H)$. Hranová množina grafu $G \boxtimes H$ je $E(G \boxtimes H) = E(G\square H) \cup E(G \times H)$.*

Příklad 9
Pro lepší pochopení zavedených pojmů, uveďme příklad všech tří operací na dvou grafech, viz Obrázek 8.



(a) Graf $P_3\square P_4$ (b) Graf $P_3 \times P_4$ (c) Graf $P_3 \boxtimes P_4$

Obrázek 8: [5] Příklady operací s grafy

Dále autoři v [5] zmiňují, že se dá dokázat, že okolí každého vrcholu v grafu $G \boxtimes H$ se dá vyjádřit vztahem

$$\forall (g,h) \in V(G \boxtimes H): \quad N_{G \boxtimes H}[(g,h)] := N_G[g] \times N_H[h], \tag{2}$$

kde symbolem $N[v]$ rozumíme $N[v] := N(v) \cup \{v\}$. Někdy se tomuto značení říká uzavřené okolí vrcholu $v \in V$. Jedno důležité upozornění je to, že v množině $N_{G \boxtimes H}[(g,h)]$ se nachází i samotný vrchol (g,h). Dá se dokázat to, že okolí libovolného vrcholu (g,h) v grafu $G \times H$, se dá vyjádřit vztahem

$$\forall (g,h) \in V(G \times H): \quad N_{G \times H}(g,h) := N_G(g) \times N_H(h). \tag{3}$$

Nyní uvedeme větu, která nám dá horní odhad chromatického čísla pro grafy, které vznikly pomocí silného součinu.

Věta 4 *[5]*
Pro jednoduché grafy G a H platí

$$\chi(G \boxtimes H) \leq \chi(G)\chi(H). \tag{4}$$

Důkaz této Věty 4 lze nalézt v [5].

3 Formulace okrajové úlohy

V této diplomové práci naší modelovou rovnicí, která v inženýrských aplikacích většinou popisuje chování fyzikální veličiny na zvoleném 3D objektu, bude Laplaceova rovnice. Uveďme základní definice a věty z funkcionální analýzy a variačního počtu.

3.1 Základní pojmy

Nejdříve si uveďme základní prostory z funkcionální analýzy. Díky tomu budeme schopni lépe popsat kvalitu nalezeného řešení Laplaceovy úlohy.

Definice 18 *[1]* ***Hilbertův prostor***
Vektorový prostor X se skalárním součinem $(\cdot,\cdot)$ se nazývá Hilbertovým prostorem, jestliže je úplný v indukované metrice ρ, tj. $\forall x, y \in X : \rho(x,y) := \|x-y\| = \sqrt{(x-y, x-y)}$.

Definice 19 *[2]* ***Sobolevův prostor***
Buď neprázdná omezená oblast $\Omega \subset \mathbb{R}^n$, kde $n \in \mathbb{N}$. Dále parametry k a p splňují $k \in \mathbb{N} \cup \{0\}$, $1 \leq p < \infty$. Pak definujeme Sobolevův prostor $W^{k,p}(\Omega)$ jako zúplnění prostoru $C^\infty(\overline{\Omega})$ vzhledem k normě

$$\|u\|_{k,p} := \left(\sum_{|\alpha| \leq k} \int_\Omega |\mathcal{D}^\alpha u(x)|^p \, dx \right)^{\frac{1}{p}}.$$

Dále si uveďme definici oblasti Ω s lipschitzovskou hranicí. Tuto definici zde uvádíme proto, aby Laplaceova úloha měla řešení. Oblast Ω bude v budoucnu reprezentovat 3D objekt, na kterém budeme řešit Laplaceovou úlohu.

Definice 20 *[2]* ***Oblast s lipschitzovskou hranicí***
Buď neprázdná a omezená oblast $\Omega \subset \mathbb{R}^n$, kde $n \in \mathbb{N} \setminus \{1\}$. Jsme-li schopni hranici oblasti Ω pokrýt konečným počtem okolí $\mathcal{U}$, kde pro každé z těchto okolí $\mathcal{U}$ existuje kartézský systém souřadnic, tj. $(y_1, y_2, \ldots, y_n) = (y', y_n)$, $\exists\, \epsilon, \delta \in \mathbb{R}^+$ a existuje funkce $a : \mathbb{R}^{n-1} \to \mathbb{R}$ tak, že platí

- $\Gamma := \mathcal{U} \cap \partial\Omega = \{(y', y_n) : \|y'\| < \delta,\ y_n = a(y')\}$,
- $\mathcal{U}^+ := \{(y', y_n) : \|y'\| < \delta,\ a(y') < y_n < a(y') + \epsilon\} \subset \Omega$,
- $\mathcal{U}^- := \{(y', y_n) : \|y'\| < \delta,\ a(y') - \epsilon < a(y') < y_n\} \subset \mathbb{R}^n \setminus \Omega$,
- $(\exists L \in \mathbb{R}^+)(\forall x', z' \in \{y' \in \mathbb{R}^{n-1} : \|y\| < \delta\}) : |a(x') - a(z')| \leq L\|x' - z'\|$.

Pak každou takovou oblast Ω nazýváme oblastí s lipschitzovskou hranici.

Poznámka 5 Zaveďme si ustálené značení, které budeme používat v této práci.

- Speciální Sobolevův prostor $W^{1,2}(\Omega)$ na oblastí Ω s lipschitzovskou hranicí budeme značit $H^1(\Omega)$, tj.

$$H^1(\Omega) := \left\{ u : \mathbb{R}^n \to \mathbb{R},\ u \in L^2(\Omega) : \frac{\partial u}{\partial x_i} \in L^2(\Omega),\ i \in \{1, 2, \ldots, n\} \right\}.$$

  Tento speciální Sobolevův prostor $H^1(\Omega)$ je taktéž Hilbertovým prostorem, díky tomu lze v tomto prostoru zavést skalární součin $\langle u, v \rangle_{H^1(\Omega)} := \langle u, v \rangle_{L^2} + \langle \nabla u, \nabla v \rangle_{L^2}$, kde $\langle u, v \rangle_{L^2} = \int_\Omega uv\, dx$.

- $H^1_\triangle(\Omega) := \{u \in H^1(\Omega) : -\triangle u \in L^2(\Omega)$ ve smyslu distribucí$\}$
- Nadále v této práci budeme hranici neprázdné, omezené oblasti $\Omega \subset \mathbb{R}^n$, tj. $\partial\Omega$, značit Γ.

Věta 5 *[2]* ***o stopách***
Nechť $\Omega \subset \mathbb{R}^3$ je neprázdná a omezená oblast s lipschitzovskou hranicí. Potom existuje právě jedno spojité a lineární zobrazení $\gamma : H^1(\Omega) \to L^2(\Gamma)$ takové, že pro každé $u \in C^\infty(\overline{\Omega})$ je $\gamma u := u|_\Gamma$. Prvek γu nazýváme stoupou funkce $u \in H^1(\Omega)$.

Definice 21 ***normovaný prostor stop***
Buď $\Omega \subset \mathbb{R}^3$ neprázdná a omezená oblast s lipschitzovskou hranicí. Pak množinu $H^{1/2}(\Gamma) := \gamma(H^1(\Omega))$ nazvěme prostorem stop.

Abychom mohli definovat duální prostor k prostoru stop, uveďme definici duálního prostoru.

Definice 22 *[1]* ***Duální prostor***
Buď X^* je normovaný lineární prostor všech lineárních funkcionálů na X, tj. $X^* := \mathcal{L}(X, \mathbb{R})$ s normou $\|A\| := \sup_{\substack{x \in X \\ \|x\| \leq 1}} |A(x)|$ nazýváme duálním prostorem k X.

Definice 23 ***duální prostor k prostoru stop***
Buď $\Omega \subset \mathbb{R}^n$ neprázdná a omezená oblast s lipschitzovskou hranicí. Prostor $H^{-1/2}(\Gamma) := \left[H^{1/2}(\Gamma)\right]^*$ je duální prostor k prostoru stop.

Dále označením γ_0 budeme značit Dirichletovu stopou, tedy zobrazení $\gamma_0 : H^1(\Omega) \to H^{1/2}(\Gamma)$. Máme-li libovolnou funkci u z prostoru $C^\infty(\overline{\Omega})$, pak označením $\gamma_0 u$ budeme rozumět jako restrikci funkce u na hranici, tj. $\gamma_0 u := u|_\Gamma$. Pomocí Dirichletovy stopy budeme značit okrajové podmínky Dirichletova typu.

Označením γ_1 budeme značit Neumannovu stopu, tedy zobrazení $\gamma_1 : H^1_\triangle(\Omega) \to H^{-1/2}(\Gamma)$. Máme-li libovolnou funkci u z prostoru $C^\infty(\overline{\Omega})$, pak označením $\gamma_1 u$, budeme rozumět jako $\frac{\partial u}{\partial n}$, kde symbol n chápeme jako jednotkový vektor vnější normály.

Poznámka 6 Dá se ukázat, že γ_0 a γ_1 jsou spojité a lineární operátory. Operátor γ_0 nám ve slabé formulaci Laplaceovy úlohy bude reprezentovat Dirichletovou okrajovou podmínku a operátor γ_1 nám bude značit Neumannovou okrajovou podmínku.

3.2 Laplaceova úloha

Laplaceova úloha nachází v inženýrských úlohách využití například při modelování proudění tepla v materiálech nebo při modelování proudění ideálních tekutin.

Definice 24 ***Laplaceova rovnice ve 3D***
Nechť $\Omega \subset \mathbb{R}^3$ je neprázdná a omezená oblast s lipschitzovskou hranicí, potom úlohu

$$\begin{cases} -\Delta u(x) = 0, & \text{pro } \forall x \in \Omega, \\ \text{okrajové podmínky,} & \text{pro } \forall x \in \Gamma, \end{cases} \tag{5}$$

nazveme Laplaceovou úlohou ve 3D.

Poznámka 7 Laplaceovým operátorem (Δ) v Definici 24 rozumíme:

$$\Delta u(x) := \sum_{i=1}^{3} \frac{\partial^2}{\partial x_i^2}.$$

Dále pak v textu budeme předpokládat, že $\Gamma_D \cup \Gamma_N = \Gamma$ a navíc $\Gamma_D \cap \Gamma_N = \emptyset$. Co se týče okrajových podmínek tak máme celkem tři druhy, které teď zmíníme.

- **Dirichletovou okrajovou podmínkou** nazveme omezení ve tvaru $\gamma_0 u(x) = g(x)$, kde $x \in \Gamma_D$ a $g \in H^{1/2}(\Gamma_D)$.
- **Neumannovou okrajovou podmínkou** nazveme omezení ve tvaru $\gamma_1 u(x) = h(x)$, kde $x \in \Gamma_N$, n je jednotkový vektor vnější normály a $h \in H^{-1/2}(\Gamma_N)$.
- **Smíšenou okrajovou podmínkou** nazveme omezení ve tvaru, kdy na množině Γ_D máme zadanou Dirichletovou okrajovou podmínku a současně na množině Γ_N máme zadanou Neumannovou okrajovou podmínku.

Uveďme fundamentální řešení Laplaceovy rovnice ve 3D, které je klíčovou součástí Věty o reprezentaci. Věta o reprezentaci je základní stavební kámen v metodě hraničních prvků pro řešení Laplaceovy rovnice. Pomocí této věty víme, jak se chová fyzikální veličina v tělese, ale nedává nám to informaci o chování funkce na hranici tělesa. Bude nezbytné zjistit, jak se chová řešení na hranici. Díky této informaci budeme schopni v dalších fázích řešit úlohu pouze na hranici 3D objektu, což je smyslem metody hraničních prvků.

Definice 25 ***Fundamentální řešení Laplaceovy rovnice***
Funkci $u^*(x,y) : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}$ *zadanou předpisem*

$$u^*(x,y) := \frac{1}{4\pi}\frac{1}{\|x-y\|} \tag{6}$$

nazveme fundamentálním řešením Laplaceovy rovnice ve 3D.

Poznámka 8 Normou v Definici 25 rozumíme eukleidovskou normu, tj. pro $x \in \mathbb{R}^3$

$$\|x\| := \sqrt{x_1^2 + x_2^2 + x_3^2}.$$

Nyní se dostáváme k bodu, kdy uvedeme Větu o reprezentaci pro Laplaceovou úlohu.

Věta 6 *[13]* ***o reprezentaci***
Buď $\Omega \subset \mathbb{R}^3$ *omezená, neprázdná oblast s lipschitzovskou hranicí a funkce* $u^*(x,y)$ *je fundamentální řešení Laplaceovy rovnice, pak pro každou funkci* $u \in H^1_{\triangle}(\Omega)$ *platí*

$$\forall \widetilde{x} \in \Omega : \quad u(\tilde{x}) := \int_\Gamma u^*(\widetilde{x},y)\gamma_1 u(y)\, dS_y - \int_\Gamma \gamma_0 u(y)\gamma_{1,y}u^*(x,y)\, dS_y. \tag{7}$$

Poznámka 9 Jednotlivé integrály z Věty 6 se nazývají potenciály. Označením $\gamma_{1,y}u^*(x,y)$ ve vzorci 7 rozumíme:

$$\gamma_{1,y}u^*(x,y) := \frac{1}{4\pi}\frac{(x-y)n(y)}{\|x-y\|^3}.$$

Definice 26 *[13] Buď* $\Omega \subset \mathbb{R}^3$ *je neprázdná a omezená oblast s lipschitzovskou hranicí. Spojitý a lineární operátor* $\widetilde{V} : H^{-1/2}(\Gamma) \to H^1(\Omega)$ *nazveme potenciál jednoduché vrstvy pro* $t \in H^{-1/2}(\Gamma)$ *a pro* $\forall \widetilde{x} \in \Omega$ *platí:*

$$(\widetilde{V}t)(\widetilde{x}) := \int_\Gamma u^*(x,y)t(y)\, dS_y. \tag{8}$$

Spojitý operátor $\widetilde{W} : H^{1/2}(\Gamma) \to H^1(\Omega)$ *nazveme potenciál dvojvrstvy pro* $s \in H^{1/2}(\Omega)$ *a pro* $\forall \widetilde{x} \in \Omega$ *platí*

$$(\widetilde{W}s)(\widetilde{x}) := \int_\Gamma \gamma_{1,y}u^*(x,y)s(y)\, dS_y. \tag{9}$$

Poznámka 10 vzorec 7 z Věty 6 lze pomocí zavedených potenciálu přepsat do následující rovnice 10, kdy za funkci t dosadíme $\gamma_1 u$ a za funkci s dosadíme $\gamma_0 u$.

$$\forall \widetilde{x} \in \Omega \quad u(\widetilde{x}) := (\widetilde{V}\gamma_1 u)(\widetilde{x}) - (\widetilde{W}\gamma_0 u)(\widetilde{x}) \tag{10}$$

Uvedený vztah platí pouze uvnitř oblasti $\Omega \subset \mathbb{R}^3$, ale neříká nic o tom, jak se řešení chová na hranici. Proto je třeba provést limitní přechody jednotlivých operátorů na hranici Γ.

Lemma 3 *[13]* ***o vlastnostech potenciálu jednoduché vrstvy***
Buď $\Omega \subset \mathbb{R}^3$ *omezená oblast s lipschitzovskou hranici a* $t \in L_\infty(\Gamma)$. *Pak potenciál* $V = \gamma_0 \circ \widetilde{V}$:

$H^{-1/2}(\Gamma) \to H^{1/2}(\Gamma)$ *je spojitý v* Ω *a pro každé* $x \in \Gamma$ *platí*

$$(Vt)(x) := \int_\Gamma u^*(x,y)t(y)\,dSy = \frac{1}{4\pi}\int_\Omega \frac{t(y)}{\|x-y\|}\,dSy.$$

Poznámka 11 Dá se dokázat, že operátor jednoduché vrstvy je omezený a $H^{-1/2}$-eliptický.

Lemma 4 *[13]* ***o vlastnostech potenciálu dvojvrstvy***
Buď $\Omega \subset \mathbb{R}^3$ *omezená oblast s lipschitzovskou hranici a* $s \in H^{1/2}(\Gamma)$*. Pak potenciál* $W = \gamma_0 \circ \widetilde{W} : H^{1/2}(\Gamma) \to H^{1/2}(\Gamma)$ *je spojitý v* Ω *pro a každé* $x \in \Gamma$ *platí*

$$(Ws)(x) := (-1 + \sigma(x))s(x) + (Ks)(x),$$

přičemž

$$(Ks)(x) := \lim_{\epsilon\to 0}\int_{y\in\Gamma:\|y-x\|\geq\epsilon} \gamma_{1,y}u^*(x,y)s(y)\,dSy = \frac{1}{4\pi}\lim_{\epsilon\to 0}\int_{y\in\Gamma:\|y-x\|\geq\epsilon}\frac{(x-y)n(y)}{\|x-y\|^3}s(y)\,dSy,$$

$$\sigma(x) := \lim_{\epsilon\to 0}\frac{1}{4\pi}\frac{1}{\epsilon^2}\int_{y\in\Omega:\|y-x\|=\epsilon} dSy.$$

Poznámka 12 Pokud je $x \in \Gamma$ na hladké části hranice pak $\sigma(x) = \frac{1}{2}$. Pokud ovšem je $x \in \Gamma$ na uzlu hranice, pak $\sigma(x)$ závisí na vnitřním úhlu.

Díky Lemmatům 3 a 4 lze Větu 6 o reprezentaci vyjádřit do tvaru, ve kterém se vyskytuje jen jedna proměnná x z hranice Γ.

$$\forall x \in \Gamma: \quad \gamma_0 u(x) = (V\gamma_1 u)(x) + \frac{1}{2}\gamma_0 u(x) - (K\gamma_0 u)(x) \tag{11}$$

Z rovnici 10, obdržíme 1. hraniční integrální rovnici pro Laplaceovou úlohu.

3.2.1 Vnitřní Dirichletova úloha

V této kapitole nastíníme řešení vnitřní Dirichletovy úlohy, která vypadá následovně.

$$\begin{cases} -\Delta u(x) = 0, & \text{pro } \forall x \in \Omega, \\ \gamma_0 u(x) = g(x) & \text{pro } \forall x \in \Gamma,\ g \in H^{1/2}(\Omega). \end{cases} \tag{12}$$

Nyní si přepíšeme 1. hraniční integrální rovnici Laplaceovy úlohy ze vzorce 11. Za operátor $\gamma_0 u$ dosadíme funkci g z Dirichletovy okrajové podmínky a upravíme. Předpokládejme, že $\forall x \in \Gamma$.

$$g(x) = (Vt)(x) + \frac{1}{2}g(x) - (Kg)(x) \tag{13}$$

Neznámou funkcí ve vnitřní Dirichletově úloze je funkce $t \in H^{-1/2}(\Gamma)$, kterou se snažíme nalézt. Hledaná funkce t musí splňovat vztah

$$\forall x \in \Gamma : \quad (Vt)(x) = \left[\frac{1}{2}I + K\right] g(x). \tag{14}$$

Poznamenejme, že symbolem $I : H^{1/2}(\Gamma) \to H^{1/2}(\Gamma)$ v rovnici 14 rozumíme operátor identity. Jedním z postupu řešení vnitřní Dirichletovy úlohy je aproximovat prostor distribucí $H^{-1/2}(\Gamma)$ konečně dimenzionálním prostorem V^n. Tento přístup v sobě obnáší volbu bázových funkcí $(\varphi_1, \varphi_2, \ldots, \varphi_n)$. Poznamenejme, že prostor V^n je možné vyjádřit pomocí lineární kombinace bázových funkcí φ_i. Pak funkci $t \in H^{-1/2}(\Gamma)$ budeme hledat v prostoru V^n.

Nyní využijeme variační postup na rovnici 14, kde obě strany rovnice vynásobíme testovací funkcí $v \in H^{-1/2}(\Gamma)$ a zintegrujeme přes hranici Γ. Tento postup se nazývá Galerkinová metoda. Za předpokladů, že potenciál jednoduché vrstvy V je omezený a $H^{-1/2}$-eliptický má rovnice 14 právě jedno řešení [13]. Obdržíme

$$(\forall v \in H^{-1/2}(\Gamma))(\forall x, y \in \Gamma) : \quad \int_\Gamma v(x)(Vt)(y)\, dSx = \int_\Gamma v(x) \left[\frac{1}{2}I + K\right] g(y)\, dSx. \tag{15}$$

Pokud se zaměříme na levou stranu rovnice 15, můžeme operátor jednoduché vrstvy $(Vt)(x)$ rozepsat na upravený tvar z Lemmatu 3 a upravit tak levou stranu rovnice 15 na tvar

$$\int_\Gamma v(x)(Vt)(x)\, dSx = \int_\Gamma v(x) \int_\Gamma \frac{1}{4\pi} \frac{t(y)}{\|x - y\|}\, dSy\, dSx. \tag{16}$$

Stejným způsobem můžeme upravit i pravou stranu rovnice 15, kde rozepíšeme operátor dvojvrstvy $(Ks)(x)$ na upravený tvar z Lemmatu 4

$$\int_\Gamma v(x) \left[\frac{1}{2}I + K\right] g(x)\, dSx = \int_\Gamma \frac{1}{2} v(x) g(x)\, dSx + \int_\Gamma v(x) \int_\Gamma \frac{1}{4\pi} \frac{(x - y)n(y)}{\|x - y\|^3} g(y)\, dSy\, dSx. \tag{17}$$

3.3 Diskretizace úlohy

V této části se zaměříme jak pomocí numerické matematiky vypočítat Laplaceovou úlohu ve 3D. Nadále v této práci budeme předpokládat, že oblast, tj. síť 3D objektu, je polygonální. Tuto oblast budeme diskretizovat pomocí triangulace. Symbolem τ_i budeme v textu značit i-tý element, tj. rovinný trojúhelník. Počet elementů sítě budeme značit F, kvůli sjednocení terminologie s teorii grafů, kde počet oblasti rovinného grafu značíme $|F|$ z anglického názvu pro oblast *face*.

$$\Gamma := \bigcup_{i=1}^{F} \overline{\tau_i} \tag{18}$$

Předtím než si uvedeme po částech lineární funkci, zaveďme si parametr sítě $h \in \mathbb{R}$

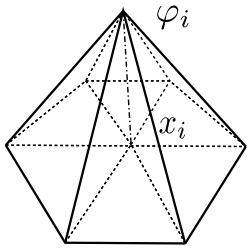
$$h := \max_i \sqrt{\text{Obsah elementu } \tau_i}. \tag{19}$$

Předpokládejme, že síť již máme rozdělenou na rovinné trojúhelníky. Symbolem N budeme značit počet uzlů sítě. Uveďme definici po částech lineární funkce, která se stane později bázovou funkcí.

Definice 27 ***po částech lineární funkce***
Pro každý bod diskretizační sítě definujeme funkci $\varphi_i : \Gamma \to \mathbb{R}$ předpisem:

$$\varphi_i(x) := \begin{cases} 1 & \text{pro } x = x_i, \text{ kde } x_i \text{ je } i\text{-tý uzel sítě}, \\ 0 & \text{pro } x = x_j \wedge x_j \neq x_i, \\ \text{jinak} & \text{afinní}. \end{cases} \tag{20}$$

kde $i, j \in \{1, 2, \ldots, N\}$. Takové funkci φ_i říkáme po částech lineární, viz Obrázek 9.



Obrázek 9: Po částech lineární bázová funkce φ_i.

Pokud se vrátíme zpět k naší vnitřní Dirichletově úloze 12, kde hledáme distribuci $t \in H^{-1/2}(\Gamma)$, můžeme funkci t aproximovat složením po částech lineárními funkcemi φ_i, tj.

$$t \approx t_h := \sum_{i=1}^{N} t_i \varphi_i,$$

kde neznámou se stávají koeficienty $t_i \in \mathbb{R}$. Po částech lineární funkce zvolíme jako bázové. Řešení můžeme reprezentovat vektorem $\mathbf{t}_h \in \mathbb{R}^N$, tj. $\mathbf{t}_h := (t_1, t_2, \ldots, t_N)^T$. Stejným způsobem vyjádřeme funkci $g \in H^{-1/2}(\Gamma)$ pomocí bázových funkcí, tj.

$$g \approx g_h := \sum_{i=1}^{F} g_i \varphi_i,$$

kde $g_i \in \mathbb{R}$. Funkci g máme zadanou ve vnitřní Dirichletové úloze 12 v Dirichletové podmínce. Pomocí L^2-projekce lze vypočítat koeficienty g_i. Tyto koeficienty g_i získáme z řešení soustavy

lineárních funkcí dané předpisem

$$\sum_{i=1}^{N} g_i \int_\Gamma \varphi_i(x)\varphi_j(x)\,dSy = \int_\Gamma g\varphi_i\,dSy, \text{ pro } j \in \{1,2,\ldots,N\}. \tag{21}$$

Nalezené koeficienty můžeme ztotožnit s vektorem $\mathbf{g}_h \in \mathbb{R}^N$, tj. $\mathbf{g}_h := (g_1, g_2, \ldots, g_N)^T$. Zaměřme se na numerické řešení rovnice 15, kterou lze v diskretizovaném tvaru zapsat pomocí matice jako

$$\mathbb{V}\mathbf{t}_h = \left(\frac{1}{2}\mathbb{M} + \mathbb{K}\right)\mathbf{g}_h, \tag{22}$$

kde matice $\mathbb{V}, \mathbb{M}, \mathbb{K}$ jsou řádu $N \times N$.

Do rovnice 16 za funkci t dosadíme funkci t_h a za funkci v dosadíme postupně funkce φ_j, kde $j \in \{1,2,\ldots,N\}$ a využijeme linearitu integrálů.

$$\int_\Gamma \varphi_j(x)(Vt_h)(x)\,dSx = \int_\Gamma \varphi_j(x)\int_\Gamma \sum_{i=1}^{N} \frac{t_i\varphi_i(y)}{4\pi\|x-y\|}\,dSy\,dSx = \sum_{i=1}^{N} t_i \int_\Gamma\int_\Gamma \frac{\varphi_j(x)\varphi_i(y)}{4\pi\|x-y\|}\,dSy\,dSx \tag{23}$$

V rovnici 23 využijeme informaci o tom, že již máme hranici Γ diskretizovanou na elementy τ_i. Definujme proto příspěvek bázových funkcí v matici $\mathbb{V}_h$ pro bázové funkce φ_i, φ_j, kde $i, j \in \{1,2,\ldots,N\}$ následovně

$$\mathbb{V}_h[j,i] := \sum_{k=1}^{F}\sum_{l=1}^{F} \int_{\tau_k}\int_{\tau_l} \varphi_j(x)\varphi_i(y)\frac{1}{4\pi}\frac{1}{\|x-y\|}\,dSy\,dSx. \tag{24}$$

Do rovnice 17 za funkci v budeme dosazovat funkce φ_j, kde $j \in \{1,2,\ldots,N\}$ a za funkci g dosadíme funkci g_h tímto způsobem.

$$\int_\Gamma \frac{1}{2}\varphi_j(x)g_h(x)\,dSx + \int_\Gamma\int_\Gamma \varphi_j(x)\frac{1}{4\pi}\frac{(x-y)n(y)}{\|x-y\|^3}g_h(y)\,dSy\,dSx =$$

$$\int_\Gamma \frac{1}{2}\varphi_j(x)\sum_{i=1}^{N} g_i\varphi_i(y)\,dSx + \int_\Gamma\int_\Gamma \varphi_j(x)\frac{1}{4\pi}\frac{(x-y)n(y)}{\|x-y\|^3}\sum_{i=1}^{N} g_i\varphi_i(y)\,dSy\,dSx =$$

$$\frac{1}{2}\sum_{i=1}^{N} g_i\int_\Gamma \varphi_j(x)\varphi_i(x)\,dSx + \sum_{i=1}^{N} g_i\int_\Gamma\int_\Gamma \varphi_j(x)\frac{1}{4\pi}\frac{(x-y)n(y)}{\|x-y\|^3}\varphi_i(y)\,dSy\,dSx. \tag{25}$$

Ve výsledku rovnice 25 využijeme informaci o tom, že již máme hranici Γ diskretizovanou na elementy τ_i. Definujeme proto příspěvky matic $\mathbb{M}_h$ a $\mathbb{K}_h$.

$$\mathbb{M}_h[j,i] := \sum_{k=1}^{F}\int_{\tau_k} \varphi_i(x)\varphi_j(y)\,dSx \tag{26}$$

$$\mathbb{K}_h[j,i] := \sum_{k=1}^{F}\sum_{l=1}^{F}\int_{\tau_k}\int_{\tau_l}\varphi_j(x)\varphi_i(y)\frac{1}{4\pi}\frac{(x-y)n(y)}{\|x-y\|^3}\,dSy\,dSx \tag{27}$$

Nyní již stačí vypočítat globální matice $\mathbb{V}, \mathbb{M}, \mathbb{K}$ a dosadit do soustavy rovnic 22. Výsledky rovnice jsou pak koeficienty t_i, pomocí kterých sestavíme funkci t_h. Funkci t_h pak dosadíme do věty o reprezentaci a získáme tak celkové řešení vnitřní Dirichletovy úlohy pro Laplaceovou rovnici ve 3D.

3.3.1 Sestavení matice $\mathbb{V}$

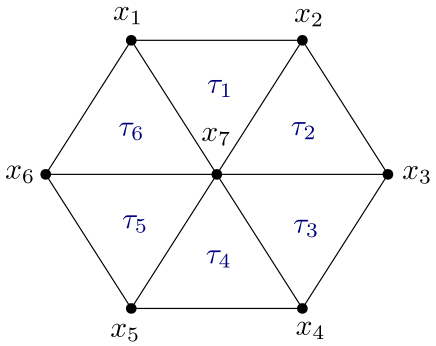
Ve vzorci 24 jsme uvedli předpis pro výpočet interakcí dvou bázových funkcí φ_i a φ_j, kde $i, j \in \{1, 2, \ldots, N\}$ v metodě hraničních prvků. Nyní si zaveďme značení $\mathbb{V}_h^{k,l}[j,i]$, pod kterým budeme rozumět výpočet jedné interakce po částech lineárních bázových funkcí $\varphi_j(x)$ a $\varphi_i(y)$ na elementech τ_k a τ_l, tj. výrazem $[j,i]$ budeme značit souřadnici v matici $\mathbb{V}$, kde je potřeba přičíst hodnotu $\mathbb{V}_h^{k,l}[j,i]$

$$\mathbb{V}_h^{k,l}[j,i] := \int_{\tau_k}\int_{\tau_l}\varphi_j(x)\varphi_i(y)\frac{1}{4\pi}\frac{1}{\|x-y\|}\,dSy\,dSx, \tag{28}$$

kde $k, l \in \{1, 2, \ldots, F\}$ a $j, i \in \{1, 2, \ldots, N\}$. Pokud je po částech lineární bázová funkce $\varphi_j : \Gamma \to \mathbb{R}$ nulová na elementu τ_k ve vzorci 28, pak výsledek interakce je nulový. Stejná situace nastává, pokud po částech lineární bázová funkce $\varphi_i : \Gamma \to \mathbb{R}$ je nulová na elementu τ_l, je pak výsledek interakce rovněž nulový. Má proto smysl počítat pouze nenulové interakce, kde bázová funkce φ_j jsou nenulové na elementu τ_k a bázová funkce φ_i je nenulová na elementu τ_j. Samotné zjištění, jaké bázové funkce φ_i a φ_j jsou nenulové na dvojici elementů τ_k a τ_l je velmi pracné. Úskalí se nachází v diskretizaci sítě, neboť každá z bázových funkcí může být nenulových nad různém počtu elementů. Jednodušší je určit jaké interakce bázových funkcí φ_i, φ_j, kde $i, j \in \{1, 2, \ldots, N\}$ jsou nenulové na konkrétní dvojicí elementů τ_k a τ_l. Uveďme Příklad 10.

Příklad 10
Uvažujme pro náš příklad již diskretizovanou síť z Obrázku 10.

Uzly sítě jsou označeny symboly $x_i \in \mathbb{R}^3$, kde $i \in \{1, 2, \ldots, 7\}$. Elementy (trojúhelníky) sítě jsou označeny symboly τ_j, kde $j \in \{1, 2, \ldots, 6\}$. Dále předpokládejme, že máme zavedené po částech lineární bázové funkce φ_i na síti z Obrázku 10, viz Definice 27. V našem příkladu nás budou zajímat interakce bázových funkcí φ_i a φ_j, které jsou nenulové na dvojicí elementů τ_1 a τ_4. Nenulovou funkční hodnotu na elementu τ_1 mají tyto bázové funkce: φ_1, φ_2 a φ_7, protože v jednom z vrcholů elementu τ_1 mají bázové funkce funkční hodnotu rovnu 1 podle Definice 27. Na elementu τ_4 jsou nenulové tyto bázové funkce: φ_4, φ_5 a φ_7. Není těžké přijít na to, že nenulový výsledek interakce na elementech τ_1 a τ_4 bude pro každé $m \in \{1, 2, 7\}$ a $n \in \{4, 5, 7\}$, tj.



Obrázek 10: Jednoduchá diskretizovaná síť

$\mathbb{V}_h^{1,4}[m,n]$. Řád lokální matice je 3×3. Na každé dvojicí trojúhelníkových elementů interagují tři trialové bázové funkce se třemi ansatz bázovými funkcemi. Pro elementy τ_1 a τ_4 sestavme lokální maticí $\mathbb{V}_{h,loc}^{1,4}$ řádu 3×3, viz Tabulka 1.

$$\mathbb{V}_{h,loc}^{1,4} = \begin{pmatrix} \mathbb{V}_h^{1,4}[1,4] & \mathbb{V}_h^{1,4}[1,5] & \mathbb{V}_h^{1,4}[1,7] \\ \mathbb{V}_h^{1,4}[2,4] & \mathbb{V}_h^{1,4}[2,5] & \mathbb{V}_h^{1,4}[2,7] \\ \mathbb{V}_h^{1,4}[7,4] & \mathbb{V}_h^{1,4}[7,5] & \mathbb{V}_h^{1,4}[7,7] \end{pmatrix}$$

Tabulka 1: Lokální matice souřadnic $\mathbb{V}_{h,loc}^{1,4}$

Nyní je potřeba těchto devět hodnot z matice $\mathbb{V}_{h,loc}^{1,4}$ přičíst do odpovídajících buněk v globální matici $\mathbb{V}_h$.

V Příkladu 10 jsme ukázali, že na libovolné dvojici trojúhelníkových elementů vzniká vždy devět nenulových interakcí, pokud zvolíme po částech lineární bázové funkce a síť bude složená pouze trojúhelníkových elementů. Pro sestavení celé matice $\mathbb{V}_h$ je potřeba sestavit celkem F^2 lokálních matic $\mathbb{V}_{h,loc}^{k,l}$, kde $k, l \in \{1, 2, \ldots, F\}$. Pro Příklad 10 by to znamenalo, že k sestavení matice $\mathbb{V}_h$ je potřeba sestavit celkem $F^2 = 7^2 = 49$ lokálních matic $\mathbb{V}_{h,loc}^{k,l}$. Navíc samotné sestavení matice $\mathbb{V}^h$ můžeme popsat sekvenčním Algoritmem 6.
Pro zvýšení efektivity a zrychlení času sestavení matice $\mathbb{V}_h$ se dá v Algoritmu 6 využít vláknový paralelismus. Ve výpočetní matematické knihovně BEM4I [11] se tento Algoritmus 6 nachází v již paralelní implementaci, kde je využita paralelní knihovna OpenMP [12]. Rozdílem oproti Algoritmu 6 je to, že před *For* smyčku, která prochází indexy elementů k, tj. 6. řádek Algoritmu 6, je vložena paralelní klauzule *#pragma paralell for schedule(8, dynamic)* a před 12. řádkem Algoritmu 6 je vložena druhá klauzule *#pragma atomic*. Důvodem, proč je před 12. řádkem Algoritmu 6 vložena klauzule, je ten, že by při paralelním běhu algoritmu mohlo dojít ke kolizím během přičtení výsledku z lokální matice $\mathbb{V}_{h,loc}$ do globální matice $\mathbb{V}_h$. Problému se vznikem kolízí se budeme zabývat v následující kapitole.

Algorithm 6 [11] Sekvenční sestavení matice $\mathbb{V}_h$ v metodě hraničních prvků

```
1: V_h = Zero2DArray[N][N]
2: V_{h,loc} = Zero2DArray[3][3]
3: kNodesIndex = Zero1DArray[3]
4: lNodesIndex = Zero1DArray[3]
5: for all element τ_k ∈ {τ} do
6:     kNodesIndex ← getNodesIndexFromElement(τ_k)
7:     for all τ_l ∈ {τ} do
8:         lNodesIndex ← getNodesIndexFromElement(τ_l)
9:         V_{h,loc} ← computeVhLoc(kNodesIndex, lNodesIndex)
10:        for all index i ∈ {1, 2, 3} do
11:            for all index j ∈ {1, 2, 3} do
12:                V_h[knodeIndex[i]][lnodeIndex[j]] += V_{h,loc}[i][j]
13:            end for
14:        end for
15:    end for
16: end for
```

4 Úvod do problému

V zadání této diplomové práce je zmíněná informace o použití teorie grafů při paralelním sestavení matic metodou hraničních prvků. My se v této práci zaměříme na sestavení matice $\mathbb{V}$, která vzniká při výpočtu parciální diferenciální rovnice s okrajovými podmínkami metodou hraničních prvků. Zaměřme se detailněji na výpočty interakcí po částech lineárních bázových funkcí na dvojicích elementů v metodě hraničních prvků. Poznamenejme, že definičním oborem těchto funkcí je celá hranice Γ, kterou jsme diskretizovali na elementy. Po částech lineární bázové funkce jsou nenulové pouze na některých elementech (trojúhelnících) sítě. Výsledek interakce bázových funkcí, tj. aproximace částečného řešení, je přičteno do příslušných buněk v globální matici $\mathbb{V}$. Buňky v matici $\mathbb{V}$ jsou jednoznačně určeny na základě očíslovaných bázových funkcí. Očíslování bázových funkcí je shodné s očíslováním uzlů sítě, neboť v daném uzlu sítě mají bázové funkce mají funkční hodnotu 1. Poznamenejme, že řád globální matice $\mathbb{V}$ je $N \times N$, kde N je počet uzlů sítě z anglického slova *Nodes*. Připomeňme, že počet po částech lineárních bázových funkcí je také N.

4.1 Kolize současných přístupů do sdílené paměti

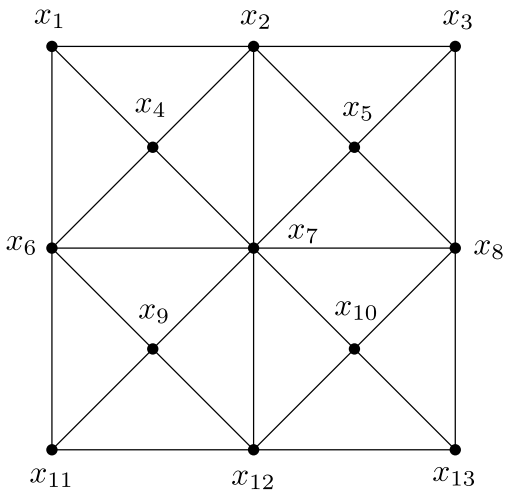
Při paralelním sestavení globální matice $\mathbb{V}$ na vícejádrové počítačové architektuře je výhodné, aby se paralelně počítaly interakce bázových funkcí na dvojicích elementů. Jedním z problémů, který tento přístup má, je případná kolize při zápisu výsledku do globální matice $\mathbb{V}$. Popišme si modelový příklad se vznikem kolizí. Mějme dvě vlákna, která ve stejný čas spočítala výsledek interakcí funkcí nad společným elementem. Chystají se přičíst svůj výsledek do stejných buněk matice $\mathbb{V}$. Nejdříve si každé vlákno přečte aktuální hodnotu uloženou v buňce a k ní přičte svůj výsledek, který zapíše do této společné buňky. Když vlákna dokončí svůj zápis do společné buňky, bude hodnota v této společné buňce matice $\mathbb{V}$ pouze posledního zapsaného vlákna. Není těžké si uvědomit, že v buňce chybí výsledek vlákna, které jako první zapsalo hodnotu do této buňky. Na straně 42 je uveden Příklad 11, který detailněji popisuje vznik kolizí.

Jedím z řešení, jak předejít tomuto nekonzistentnímu stavu v buňkách matice $\mathbb{V}$, je označit tuto operaci, tj. přičtení lokální matice do globální matice $\mathbb{V}$, jako tzv. atomickou operaci. V paralelní aplikaci, ve které je kolizní operace označená jako atomická operací, máme zaručeno, že jednotlivá vlákna mohou bezpečně zapisovat do sdílené paměti. V našem případě se jedná o zápis do buněk v matici $\mathbb{V}$. Většinou je tento mechanismus implementován s podporou na hardwarové vrstvě (CPU architektury). Existují i počítačové architektury, na kterých sice je implementovaná podpora vykonávaní atomické operace ale s jistými omezeními, například synchronizace ve stejném bloku vláken ale ne však synchronizace se všemi paralelními vlákny. Jistým zástupcem takové oblasti jsou GPU architektury. Samotná softwarová a hardwarová podpora vykonávání

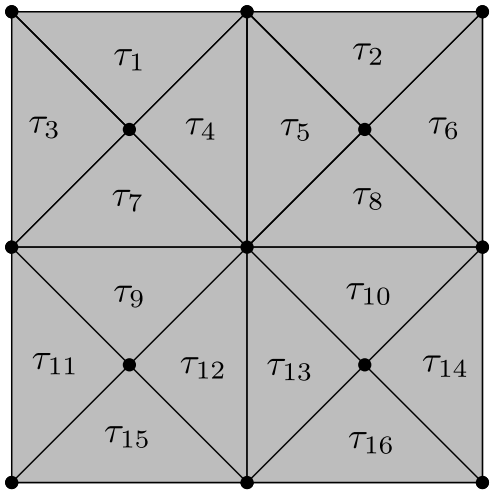
atomické operace nese větší systémové nároky na kontrolu zapisovaného místa, protože musí ošetřit stav, kdy některá vlákna chtějí zapsat výsledek do stejné paměti. V této diplomové práci se zaměříme na alternativní postup, jak vyřešit kolize vláken při zápisu do globální matice $\mathbb{V}_h$. Nebudeme využívat atomické operace, místo toho se pokusíme náš problém s kolizemi vyřešit za pomocí dobrého vrcholového barvení grafu. Jednotlivé barvy mohou např. označovat časové sloty, kdy dvojice elementů obarveny stejnou barvou, mohou zapisovat do různých buněk v globální matici $\mathbb{V}$ v pořadí, při kterém nevznikají kolize.

Příklad 11

V našem příkladu budeme sestavovat matici $\mathbb{V}$. Obdobně jako v Příkladu 10 ze strany 38 se budeme se zabývat vnitřní Dirichletovou úlohou Laplaceovy rovnice ve 3D. Zaměřme se na problém s kolizemi při zápisu hodnot z lokálních matic $\mathbb{V}_{loc}^{k,l}$ do globální matice $\mathbb{V}$ během paralelního vykonávání, kde k, l jsou indexy elementů τ. Pro jednoduchost uvažujme jednoduchou hranici 3D objektu, která je rozdělená diskretizací na elementy (trojúhelníky), viz Obrázek 11. Ve výpočtech budeme uvažovat po částech lineární bázové funkce. Z Obrázku 11(a) je vidět, že síť má celkem 13 uzlů x_i, kde $i \in \{1, 2, \ldots, 13\}$. Předpokládejme, že stejný počet je i po částech lineárních bázových funkcí φ_i, které jsou definované na této síti. Síť je rozdělená na 16 elementů (trojúhelníků), viz Obrázek 11(b). Elementy jsme označili symboly τ_j, kde $j \in \{1, 2, ..., 16\}$. Připomeňme si, že řád globální matice $\mathbb{V}$ z Obrázku 11 je 13×13.



(a) Označené body sítě



(b) Označené elementy sítě

Obrázek 11: Příklad 2D hranice 3D objektu

Během sestavení globální matice $\mathbb{V}$ je zapotřebí vypočítat všechny lokální matice $\mathbb{V}_{loc}^{k,l}$, kde $k, l \in \{1, 2, \ldots, 6\}$. Počet lokálních matice je v našem modelovém případě $16^2 = 256$. Poznamenejme, že při sestavení každé lokální matice $\mathbb{V}_{loc}^{k,l}$ se vždy počítá celkem devět interakcí šesti

bázových funkcí.

V následujícím modelovém příkladu budeme počítat interakce bázových funkcí na dvojicí elementů (τ_1, τ_7) a (τ_3, τ_4). Připomeňme, že vždy existují tři bázové funkce φ_i, které mají nenulovou funkční hodnotu definovanou nad elementem τ_j. (Každá z těchto tří po částech lineárních bázových funkcí má funkční hodnotu rovnu jedničce v jednom ze tří vrcholů trojúhelníku. V ostatních vrcholech sítě jsou tyto bázové funkce nulové.) Sestavme lokální matice $\mathbb{V}_{loc}^{1,7}$ a $\mathbb{V}_{loc}^{3,4}$ stejným způsobem jako v Příkladu 10. Výrazem $\mathbb{V}^{k,l}[j,i]$ v této kapitole budeme rozumět

$$\mathbb{V}^{k,l}[j,i] := \int_{\tau_k}\int_{\tau_l} \varphi_j(x)\varphi_i(y)\frac{1}{4\pi}\frac{1}{\|x-y\|}\,dSy\,dSx.$$

stejně jako výrazu $\mathbb{V}_h^{k,l}[j,i]$ ve vzorci 28 ze strany 38. Připomeňme, že výrazem $[j,i]$ je určen index v matici $\mathbb{V}$ do kterého se přičte hodnota $\mathbb{V}_h^{k,l}[j,i]$. Nyní již uvedeme lokální matice $\mathbb{V}_{loc}^{1,7}$ a $\mathbb{V}_{loc}^{3,4}$. Popis vzniku lokálních matic $\mathbb{V}_{loc}^{k,l}$ je detailně popsán v Příkladu 10 na straně 38.

$$\mathbb{V}_{loc}^{1,7} = \begin{pmatrix} \mathbb{V}^{1,7}[1,4] & \mathbb{V}^{1,7}[1,6] & \mathbb{V}^{1,7}[1,7] \\ \mathbb{V}^{1,7}[2,4] & \mathbb{V}^{1,7}[2,6] & \mathbb{V}^{1,7}[2,7] \\ \mathbb{V}^{1,7}[4,4] & \mathbb{V}^{1,7}[4,6] & \mathbb{V}^{1,7}[4,7] \end{pmatrix}$$

Tabulka 2: Lokální matice souřadnic $\mathbb{V}_{loc}^{1,7}$

$$\mathbb{V}_{loc}^{3,4} = \begin{pmatrix} \mathbb{V}^{3,4}[1,2] & \mathbb{V}^{3,4}[1,4] & \mathbb{V}^{3,4}[1,7] \\ \mathbb{V}^{3,4}[4,2] & \mathbb{V}^{3,4}[4,4] & \mathbb{V}^{3,4}[4,7] \\ \mathbb{V}^{3,4}[6,2] & \mathbb{V}^{3,4}[6,7] & \mathbb{V}^{3,4}[6,7] \end{pmatrix}$$

Tabulka 3: Lokální matice souřadnic $\mathbb{V}_{loc}^{3,4}$

Pro jednoduchost předpokládejme, že matice $\mathbb{V}$ bude sestavena paralelně pomocí dvou paralelních vláken *thread#1* a *thread#2*. Vlákno *thread#1* bude sestavovat lokální matici $\mathbb{V}_{loc}^{1,7}$ a vlákno *thread#2* bude sestavovat lokální matici $\mathbb{V}_{loc}^{3,4}$. Ve stejnou chvíli obě vlákna sestaví příslušné lokální matice a rozhodnou se lokální příspěvky ze svých lokálních matic zapsat do příslušných buněk v globální matici $\mathbb{V}$. Pokud v kódu není přičtení lokálních příspěvků do globální matice $\mathbb{V}$ ošetřeno např. atomickou operací, může pak nastat kolize. Z Tabulek 2 a 3 lze vyčíst, že se budou některé lokální příspěvky přičítat do společných buněk v matici $\mathbb{V}$. Jedná se o tyto buňky v matici $\mathbb{V}$: $\mathbb{V}[1,4], \mathbb{V}[1,7], \mathbb{V}[4,4]$ a $\mathbb{V}[4,7]$. Označme si symbolem $y \in \mathbb{R}$ hodnotu buňky $\mathbb{V}[1,4]$, tj. $y = V[1,4]$, v okamžiku kdy vlákna *thread#1* a *thread#2* začínají sestavovat své lokální matice $\mathbb{V}_{loc}^{1,7}$ a $\mathbb{V}_{loc}^{3,4}$. Pokud by ve stejný čas vlákno *thread#1* chtělo přičíst hodnotu $\mathbb{V}^{1,7}[1,4]$ do buňky $\mathbb{V}[1,4]$ stejně tak jako druhé vlákno *thread#2* hodnotu $\mathbb{V}^{3,4}[1,4]$, potom hodnota buňky $\mathbb{V}[1,4]$ nebude daná součtem $y + \mathbb{V}^{1,7}[1,4] + \mathbb{V}^{3,4}[1,4]$, ale pouze $y + \mathbb{V}^{1,7}[1,4]$, nebo $y + \mathbb{V}^{3,4}[1,4]$. To, jaká hodnota v buňce $\mathbb{V}[1,4]$ nakonec zůstane, bude záležet na tom, které z vláken zapsalo svůj výsledek do této buňky $\mathbb{V}[1,4]$ jako poslední. Situaci, kterou jsme popsali, budeme nazývat kolizí. Kolize by mohla nastat i pro zbylé společné buňky v matici $\mathbb{V}$, tj. $\mathbb{V}[1,7], \mathbb{V}[4,4]$ a $\mathbb{V}[4,7]$.

4.2 Matice počtu zápisu do jednotlivých buněk v matici $\mathbb{V}$

V této kapitole se podíváme na počty přístupu k jednotlivým buňkám během sestavení matice $\mathbb{V}$. K samotnému určení počtu zápisů využijeme sekvenční Algoritmus 6 ze strany 40. Abychom odlišili matici $\mathbb{V}$ od nové matice zápisu, nazvěme proto novou matici zápisu $\mathbb{V}'$. Nová matice $\mathbb{V}'$ bude stejného řádu jako matice $\mathbb{V}$. Hodnoty v buňkách matice $\mathbb{V}'$ budou odpovídat informacím o počtech zápisu lokálních výsledků do buněk v matici $\mathbb{V}$. K sestavení matice $\mathbb{V}'$ bude potřeba provést pár úprav v Algoritmu 6. První úprava Algoritmu 6 se týká jmenné konvence na 1. řádku, kde matici $\mathbb{V}_h$ přejmenujeme na $\mathbb{V}'$. Dále 2. a 8. řádek Algoritmu 6 zakomentujeme, protože nebude potřeba provádět operace s lokální maticí $\mathbb{V}_{h,loc}$. Poslední úprava se týká na 12. řádku Algoritmu 6, kdy matici $\mathbb{V}_h$ nahradíme maticí $\mathbb{V}'$ a místo hodnoty $\mathbb{V}_{h,loc}[i][j]$ dáme jedničku. Přičtení jedničky na 12. řádku upraveného Algoritmu 6 má význam přičtení jedničky za zápis do buňky $\mathbb{V}$[knodeIndex[i]][lnodeIndex[j]]. Buňky v matici $\mathbb{V}'$ se chovají jako čítače.

Pro Příklad 11 ze strany 42 by taková matice $\mathbb{V}'$ vypadala následovně: viz Tabulka 4. Je nutné zmínit, že hodnoty v Tabulce 4 závisí na původní geometrií sítě.

$$\mathbb{V}' = \begin{array}{c} \\ \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \mathbf{5} \\ \mathbf{6} \\ \mathbf{7} \\ \mathbf{8} \\ \mathbf{9} \\ \mathbf{10} \\ \mathbf{11} \\ \mathbf{12} \\ \mathbf{13} \end{array} \begin{array}{c} \begin{array}{cccccccccccc} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} & \mathbf{8} & \mathbf{9} & \mathbf{10} & \mathbf{11} & \mathbf{12} & \mathbf{13} \end{array} \\ \begin{pmatrix} 4 & 8 & 4 & 8 & 8 & 8 & 16 & 8 & 8 & 8 & 4 & 8 & 4 \\ 8 & 16 & 8 & 16 & 16 & 16 & 32 & 16 & 16 & 16 & 8 & 16 & 8 \\ 4 & 8 & 4 & 8 & 8 & 8 & 16 & 8 & 8 & 8 & 4 & 8 & 4 \\ 8 & 16 & 8 & 16 & 16 & 16 & 32 & 16 & 16 & 16 & 8 & 16 & 8 \\ 8 & 16 & 8 & 16 & 16 & 16 & 32 & 16 & 16 & 16 & 8 & 16 & 8 \\ 8 & 16 & 8 & 16 & 16 & 16 & 32 & 16 & 16 & 16 & 8 & 16 & 8 \\ 16 & 32 & 16 & 32 & 32 & 32 & 64 & 32 & 32 & 32 & 16 & 32 & 16 \\ 8 & 16 & 8 & 16 & 16 & 16 & 32 & 16 & 16 & 16 & 8 & 16 & 8 \\ 8 & 16 & 8 & 16 & 16 & 16 & 32 & 16 & 16 & 16 & 8 & 16 & 8 \\ 8 & 16 & 8 & 16 & 16 & 16 & 32 & 16 & 16 & 16 & 8 & 16 & 8 \\ 4 & 8 & 4 & 8 & 8 & 8 & 16 & 8 & 8 & 8 & 4 & 8 & 4 \\ 8 & 16 & 8 & 16 & 16 & 16 & 32 & 16 & 16 & 16 & 8 & 16 & 8 \\ 4 & 8 & 4 & 8 & 8 & 8 & 16 & 8 & 8 & 8 & 4 & 8 & 4 \end{pmatrix} \end{array}$$

Tabulka 4: Matice počtu zápisu $\mathbb{V}'$ do jednotlivých buněk v matici $\mathbb{V}$

Nyní odvodíme vzorec výpočtu pro počet zápisů do jednotlivých buněk v globální matici $\mathbb{V}$. K tomu nám pomůže nové zobrazení V_τ, které si nyní definujeme.

Definice 28 *Mějme diskretizovanou síť na uzly x_i a elementy τ_j, kde $i \in \{1, 2, \ldots, N\}$ a $j \in \{1, 2, \ldots, F\}$. Množinu všech uzlů sítě označme X a množinu všech elementů τ. Pak zobrazení*

$I : X \times \tau \to \{0, 1\}$ *nazveme funkcí incidence, kterou definujeme vztahem*

$$I(x_i, \tau_j) := \begin{cases} 1, & \textit{pro } x_i \in \tau_j \quad \textit{uzel } x_i \textit{ je vrcholem elementu } \tau_j, \\ 0, & \textit{pro } x_i \notin \tau_j \quad \textit{vrchol } x_i \textit{ není vrcholem elementu } \tau_j. \end{cases}$$

Dále zobrazení $V_\tau : X \to \mathbb{N} \cup \{0\}$ *nazveme počtem sousedních elementů k zadanému uzlu* x_i *a bude mít předpis*

$$V_\tau(x_i) := \sum_{j=1}^{F} I(x_i, \tau_j)$$

Poznámka 13 Definici 28 lze ekvivalentně zavést i pro rovinné grafy, kde by množina uzlů byla nahrazena množinou vrcholů a množina elementů by byla nahrazena množinou oblastí. Symbolem $\mathbb{N}_0$ budeme značit množinu $\mathbb{N} \cup \{0\}$. Pro Příklad 11 uveďme tabulku vypočtených hodnot V_τ.

i	1	2	3	4	5	6	7	8	9	10	11	12	13
$V_\tau(x_i)$	2	4	2	4	4	4	8	4	4	4	2	4	2

Tabulka 5: Vypočtené hodnoty V_τ

Hodnotu $V_\tau(x_7) = 8$, protože uzel x_7 je vrcholem celkem osmi elementů, tj. $\{\tau_4, \tau_5, \tau_7, \tau_8, \tau_9, \tau_{10}, \tau_{11}, \tau_{12}, \tau_{13}\}$, ale například $V_\tau(x_1) = 2$, neboť uzel x_1 je vrcholem pouze elementů τ_1 a τ_3. Nyní již máme nachystaný nezbytný aparát k tomu, abychom zavedli vzorec pro výpočet přístupů k jednotlivým buňkám v globální matici $\mathbb{V}$, který vypadá následovně

$$\mathbb{V}'[j, i] := V_\tau(x_j) \cdot V_\tau(x_i). \tag{29}$$

Indexem j značíme řádek a indexem i značíme sloupec matice $\mathbb{V}'$. Například $\mathbb{V}'[7, 7] = V_\tau(x_7) \cdot V_\tau(x_7) = 8 \cdot 8 = 64$, což vidíme, že se hodnota shoduje s buňkou $\mathbb{V}'[7][7]$ v Tabulce 4. Pokud bychom vytvořili vektor $\mathbf{V}_\tau \in \mathbb{N}_0^N$, který by pro náš případ vypadal následovně
$\mathbf{V}_\tau = (2, 3, 2, 4, 4, 4, 8, 4, 4, 4, 2, 4, 2) \in \mathbb{N}_0^{13}$, pak bychom mohli matici $\mathbb{V}'$ vyjádřit vztahem

$$\mathbb{V}' = \mathbf{V}_\tau^T \cdot \mathbf{V}_\tau. \tag{30}$$

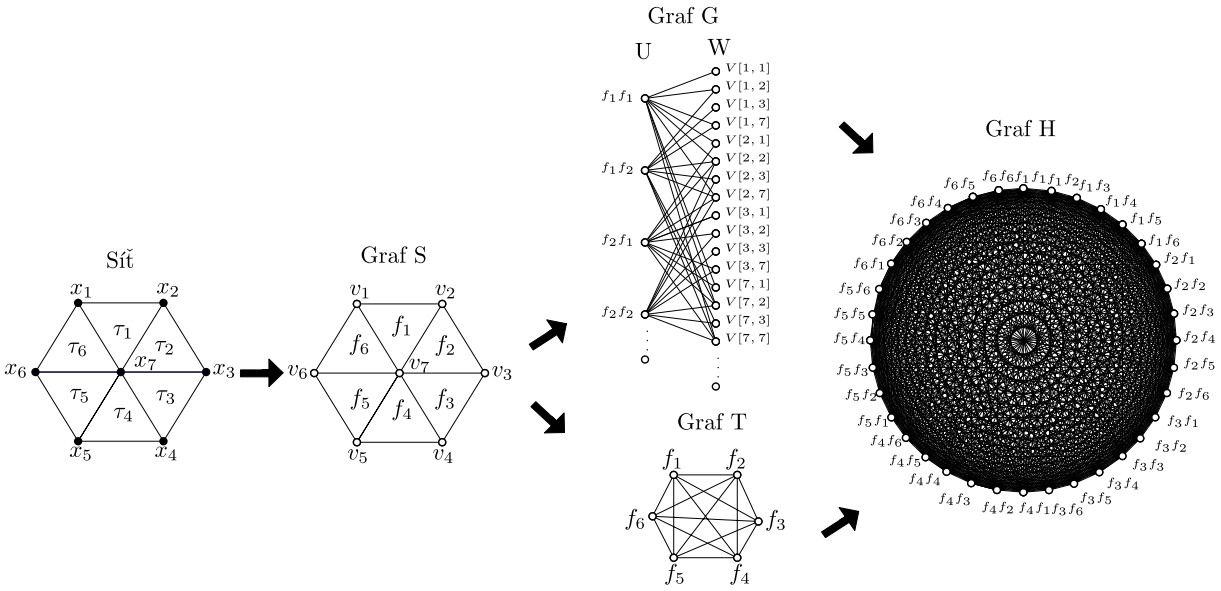
Symbolem T v rovnici 30 značíme operaci transpozice prováděnou nad vektorem. Z Tabulky 4 vyplývá, že je zapotřebí použít alespoň 64 různých časových množin při paralelním sestavení matice $\mathbb{V}$ pro zadanou síť z Obrázku 11. V našem Příkladu 11 se 64 krát přistupuje k buňce $\mathbb{V}[7, 7]$ viz. Tabulka 4. Žádná z těchto 64 lokálních matic $\mathbb{V}_{loc}^{k,l}$, kde $k, l \in \{4, 5, 8, 9, 10, 11, 12, 13\}$, nemůže být paralelně přičítána do globální matice $\mathbb{V}$ s žádnou jinou lokální matici, neboť by mohlo dojít ke kolizi při neošetření zápisu do matice $\mathbb{V}$ například pomocí atomické operace.

4.3 Návrh řešení

Dostáváme k samotnému návrhu. Cílem je vytvořit algoritmus, který by rozdělil dvojice elementů sítě do speciálních disjunktních množin. Elementy ze stejné množiny by mohly být vykonávány paralelně a ve stejném časovém slotu bez toho, aby nastala kolize při zápisech hodnot z lokálních matic do globální matice $\mathbb{V}$. Důležitou podmínkou, kterou klademe na algoritmus je to, aby rychle rozdělil elementy do těchto disjunktních množin. Časovou náročnost, kterou bychom si přáli u algoritmu dosáhnout, je $\mathcal{O}(n)$, kde n je počet interagujících dvojic elementů. Je-li F počet elementů diskretizované sítě, pak $n = F^2$. Důvodem, proč chceme dosáhnout právě takové časové náročnosti, je to, že budoucí algoritmus nejspíš bude muset přistoupit ke každé dvojici elementů a určit příslušnost dvojice elementů k dané disjunktní množině. Počet bázových funkcí se v metodě hraničních prvků pohybuje okolo 10^3 až 10^4 [14] u *fast BEM* lze dosáhnout i 10^6 bázových funkcí [14]. Počet elementů sítě, která se dá vyjádřit rovinným grafem, můžeme v metodě hraničních prvků odhadnout dvojnásobkem počtu uzlů, viz Důsledek 1 ze strany 19. Počet vstupních dvojic elementů v budoucím algoritmu nám vychází v rozmezí 10^6 až 10^{12}, což není zrovna malé číslo.

4.4 Využití teorie grafů

V úvodu této kapitoly 4.4.1 zmíníme, jak z původní sítě 3D modelu vytvoříme graf S. Po té v sekci 4.4.2 popíšeme konstrukci nového bipartitního grafu G za pomocí grafu S. Graf G bude popisovat zápis hodnot z lokálních matic $\mathbb{V}_{loc}^{k,l}$ do buněk v matici $\mathbb{V}$, navíc graf G bude bipartitní. Vrcholy v partitě U grafu G budou dvojice oblastí f_k, f_l (element τ_k budeme v grafovém pojetí nazývat oblastí f_k a analogicky element τ_l budeme v grafovém pojetí nazývat oblast f_l), symbolizující příslušný výpočet lokální matice $\mathbb{V}_{loc}^{k,l}$. Ve druhé partitě W grafu G se vrcholy stanou jednotlivé buňky matice $\mathbb{V}$. V sekci 4.4.3 uvedeme popis grafu H, který vznikne z grafu G. Graf H bude popisovat situaci, kdy dvě lokální matice $\mathbb{V}_{loc}^{k,l}$ přistupují ke stejným buňkám v globální matici $\mathbb{V}$ během sestavení matice $\mathbb{V}$ v metodě hraničních prvků. V kapitole 4.4.2 popíšeme vznik nového grafu T, který nám řekne, že graf H je dán operací silného součinu dvou grafů T. V závěru této kapitoly zmíníme dobré vrcholové barvení grafu H, pomocí kterého půjde vyřešit problém s kolizemi bez využití atomické operace. Tím, že stanovíme hledané disjunktní množiny pomocí barevných tříd grafu H.



Obrázek 12: Schéma konstrukce grafů

4.4.1 Popis grafu S

V této malé podkapitole popíšeme vznik grafu S ze sítě 3D objektu. Uzly sítě x_i prohlásíme za vrcholy v_i grafu S. Úsečky, které propojují jednotlivé uzly sítě, se stanou hrany v grafu S. Pokud bychom se podívali na síť a k této síti vytvořili odpovídající rovinný graf S, tak zjistíme, že grafický popis sítě a grafu S je identický. Elementy sítě τ_j budeme v rovinném případu nazývat oblastmi f_j grafu S. Ve schématu z Obrázku 12 vidíme, že uzly sítě x_i odpovídají vrcholům v_i v grafu S, pro $i \in \{1, 2, \ldots, 7\}$. Elementům sítě τ_j v Obrázku 12 odpovídají oblasti f_j grafu S, kde $j \in \{1, 2, \ldots, 6\}$.

4.4.2 Popis grafu G

V Příkladu 11 jsme nastínili problém s kolizemi. Předpokládejme, že již máme vytvořený graf S k dané síti. Následně se zaměřme se na sestavení matice $\mathbb{V}$. Danou situaci můžeme popsat bipartitním grafem $G = (V, E)$, kde vrcholovou množinu V tvoří dvě disjunktní množiny U a W. Partitu U budou tvořit dvojice oblastí grafu S, kterých je v Příkladu 11 celkem $16^2 = 256$ (rozlišujeme pořadí oblastí ve dvojicích). Dvojice oblastí (elementů) (f_i, f_j), tj. vrcholy v množině U, mají v našem grafovém pojetí význam vypočtené lokální matice $V_{loc}^{k,l}$. Druhou partitu W budou tvořit vrcholy reprezentující jednotlivé buňky v globální matici $\mathbb{V}$. Množina W by pro Příklad 11 obsahovala $13^2 = 169$ vrcholů. Z každého vrcholu množiny U odchází celkem devět hran, protože výsledky z lokální matice $\mathbb{V}_{loc}^{k,l}$ se přičítají do devíti buněk v matici $\mathbb{V}$. Hrany z vrcholové množiny U vedou pouze do vrcholové množiny W grafu G, pokud by tomu tak nebylo, graf G by nebyl bipartitní. Každá hrana v tomto grafu má význam zápisu hodnoty

buňky z lokální matice $\mathbb{V}_{loc}^{k,l}$ do dané buňky v globální matici $\mathbb{V}$. Snadno spočítáme počet hran grafu G. Graf G by v Příkladu 11 obsahoval

$$|E| = 9|U| = 9 \cdot 256 = 2304 \text{ hran},$$

$$|V| = |U| + |W| = 169 + 256 = 425 \text{ vrcholů}.$$

Poznamenejme, že jednotlivé stupně vrcholů z množiny W v Příkladu 11 odpovídají hodnotám v matici $\mathbb{V}'$, viz Tabulka 4. Pokud bychom chtěli řešit daný problém s kolizemi za použití grafu G, bylo by zapotřebí najít nezávislé množiny rozkladu množiny U, ve kterých by v každé množině byly pouze ty dvojice oblastí, které neobsahují žádnou společnou souřadnici ve své lokální matici souřadnic. V anglické literatuře se tento pojem nazývá *exact cover* množiny U.

4.4.3 Popis grafu H

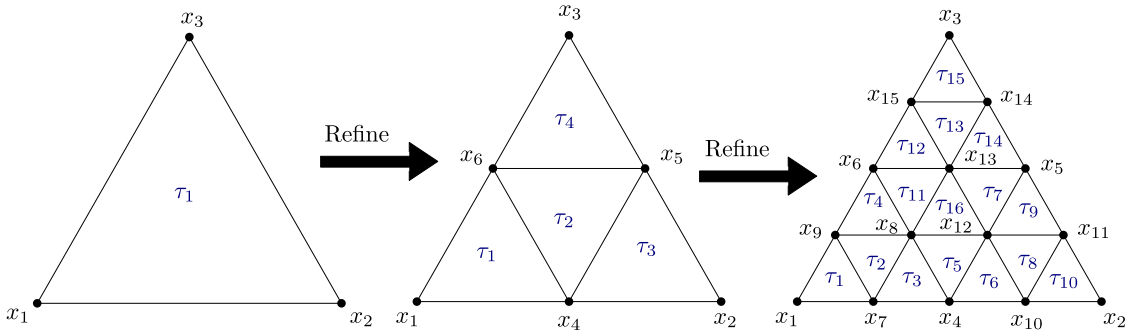
My se však v této práci nebude zaměřovat na *exact cover*, ale na dobré vrcholové barvení grafu, které je alternativní možností, jak je možné vyřešit problém s kolizemi. Budeme konstruovat nový graf $H = (U, E')$, jehož vrcholová množina U je totožná s vrcholovou množinou U grafu G. Opět i zde vrcholy z množiny U reprezentují dvojice elementů f_k, f_l $\left(\text{výpočet lokální matice } \mathbb{V}_{loc}^{k,l}\right)$. Hrana mezi dvěma vrcholy z množiny U grafu H bude existovat právě tehdy, jestliže tyto dva vrcholy z vrcholové množiny U grafu G mají alespoň jeden společný sousední vrchol v množině W grafu G. Hranami v grafu H popisujeme vlastnost, že dané dvě sestavené lokální matice $\mathbb{V}_{loc}^{k,l}$ mají při zápisu svých lokálních příspěvků do globální matice $\mathbb{V}$ alespoň jednu společnou buňku v matici $\mathbb{V}$. Dvojice vrcholů z množiny U grafu H, které jsou spojené hranou, nemohou být vykonávané paralelně ve stejném časovém okamžiku. Graf H z Příkladu 11 má $|U| = |F|^2 = |16|^2 = 256$ vrcholů, což je méně vrcholů, než je v grafu G. Co se týká počtu hran grafu H tak, těch může být méně než v původním grafu G, ale určitě ne více než 2304 hran.

4.4.4 Dobré vrcholové barvení grafu H

V úvodu této kapitoly jsme nastínili problém s kolizemi při paralelním sestavení matice $\mathbb{V}$. Připomeňme, že každé vlákno sestavuje jiné lokální matice $\mathbb{V}_{loc}^{k,l}$. Vrcholy grafu G a H odpovídají jednotlivým dvojicím oblastí grafu S resp. elementům sítě, na kterých je sestavena příslušná lokální matice $\mathbb{V}_{loc}^{k,l}$ v metodě hraničních prvků. Hrany v grafu G nám modelují zápis lokálních příspěvků z lokální matice $\mathbb{V}_{loc}^{k,l}$ do buněk v globální matici $\mathbb{V}$. Oproti tomu hrana v grafu H nám reprezentuje kolizi při paralelním běhu mezi zápisem dvou lokálních matic $\mathbb{V}_{loc}^{k,l}$ do globální matice $\mathbb{V}$. Našim úkolem je rozdělit množinu všech dvojic oblastí (elementů) na jednotlivé disjunktní podmnožiny tak, že pro všechny dvojice oblastí (elementů) ze stejné množiny bude zaručeno, že zápisy hodnot z lokálních matic do matice $\mathbb{V}$ bude moct být prováděn paralelně

bez vzniku kolizí. K určení takových množin využijeme dobré vrcholové barvení grafu H. Víme, že na okolí každého vrcholů, který má barvu b_i, už žádný sousední vrchol barvu b_i mít nemůže. Pokud by sousední vrchol libovolného vrcholu v grafu H měl stejnou barvu, znamenalo by to kolizi, neboť Graf H je grafem kolizí. Jednotlivé barvy v dobrém barvení grafu H odpovídají časovým slotům v paralelním běhu program. Obarvené vrcholy grafu H stejnou barvou, tj. zápisy hodnot z lokálních matic do buněk v matici $\mathbb{V}$, může být vykonáno paralelně, neboť je zde zaručeno, že nemůže nastat kolize.

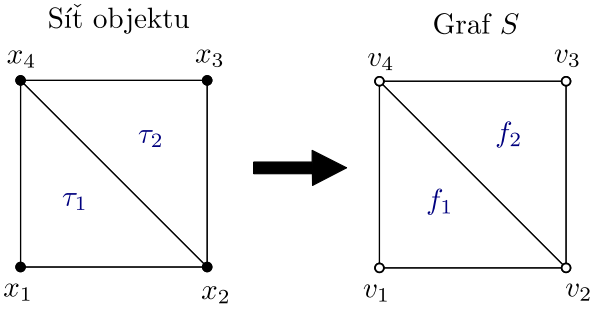
Poznámka 14 Předpokládejme od této chvíle, že síť, kterou je pokryté trojrozměrné těleso bude mít trojúhelníkový vzor, viz Obrázek 13. S tím je spojeno i to, že graf S vytvořený k této síti, bude graficky identický se svou sítí. Důvodem, proč takovou síť zmiňujeme, je fakt, že taková síť bývá výstupem z různých programů, při zjemnění sítě. Setkat se s takovou sítí můžeme i v knihovně BEM4I [11]. Stačí, aby se provedlo dvojí zjemnění a nově vzniklé uzly uvnitř trojúhelníku, budou vrcholy šesti elementů (trojúhelníku) např. uzly x_8, x_{12} a x_{13} z Obrázku 13.



Obrázek 13: Trojúhelníkovou síť - ukázka zjemnění

Příklad 12

Pro lepší pochopení předchozích tří grafů S, G a H uvádíme nejjednodušší příklad a to síť, která je tvořena dvěma elementy τ_1 a τ_2 a čtyřmi uzly x_i, kde $i \in \{1, 2, \ldots, 4\}$. Naše síť je zobrazena na Obrázku 14.



Obrázek 14: Síť a odpovídající graf S k síti

Nyní pro síť z Obrázku 14 sestavíme graf S, který ze zadaných bodů sítě vytvoří jednotlivé vrcholy a z hran elementů vytvoří hrany grafu S. Elementy sítě se stanou oblastmi grafu S, viz Obrázek 14. Opět i v tomto příkladu uvažujme čtyři po částech lineární bázové funkce φ_i. Protože máme pouze dva elementy sítě τ_1 a τ_2 bude potřebné sestavit právě čtyři lokální matice $\mathbb{V}_{loc}^{k,l}$, kde $k, l \in \{1, 2\}$. Poznamenejme, že matice $\mathbb{V}$ bude řádu 4×4. Uveďme si všechny čtyři lokální matice $\mathbb{V}_{loc}^{k,l}$. Taktéž můžeme uvést matici $\mathbb{V}'$. Dříve než tak učiníme, vytvořme se Ta-

$$\mathbb{V}_{loc}^{1,1} = \begin{pmatrix} \mathbb{V}^{1,1}[1,1] & \mathbb{V}^{1,1}[1,2] & \mathbb{V}^{1,1}[1,4] \\ \mathbb{V}^{1,1}[2,1] & \mathbb{V}^{1,1}[2,2] & \mathbb{V}^{1,1}[2,4] \\ \mathbb{V}^{1,1}[4,1] & \mathbb{V}^{1,1}[4,2] & \mathbb{V}^{1,1}[4,4] \end{pmatrix}$$

Tabulka 6: Lokální matice souřadnic $\mathbb{V}_{loc}^{1,1}$

$$\mathbb{V}_{loc}^{1,2} = \begin{pmatrix} \mathbb{V}^{1,2}[1,2] & \mathbb{V}^{1,2}[1,3] & \mathbb{V}^{1,2}[1,4] \\ \mathbb{V}^{1,2}[2,2] & \mathbb{V}^{1,2}[2,3] & \mathbb{V}^{1,2}[2,4] \\ \mathbb{V}^{1,2}[4,2] & \mathbb{V}^{1,2}[4,3] & \mathbb{V}^{1,2}[4,4] \end{pmatrix}$$

Tabulka 7: Lokální matice souřadnic $\mathbb{V}_{loc}^{1,2}$

$$\mathbb{V}_{loc}^{2,1} = \begin{pmatrix} \mathbb{V}^{2,1}[2,1] & \mathbb{V}^{1,1}[2,2] & \mathbb{V}^{2,1}[2,4] \\ \mathbb{V}^{2,1}[3,1] & \mathbb{V}^{1,1}[3,2] & \mathbb{V}^{2,1}[3,4] \\ \mathbb{V}^{2,1}[4,1] & \mathbb{V}^{1,1}[4,2] & \mathbb{V}^{2,1}[4,4] \end{pmatrix}$$

Tabulka 8: Lokální matice souřadnic $\mathbb{V}_{loc}^{2,1}$

$$\mathbb{V}_{loc}^{2,2} = \begin{pmatrix} \mathbb{V}^{2,2}[2,2] & \mathbb{V}^{1,2}[2,3] & \mathbb{V}^{2,2}[2,4] \\ \mathbb{V}^{2,2}[3,2] & \mathbb{V}^{1,2}[3,3] & \mathbb{V}^{2,2}[3,4] \\ \mathbb{V}^{2,2}[4,2] & \mathbb{V}^{1,2}[4,3] & \mathbb{V}^{2,2}[4,4] \end{pmatrix}$$

Tabulka 9: Lokální matice souřadnic $\mathbb{V}_{loc}^{2,2}$

bulku 10, kde pro zadané uzly sítě x_i spočtěme hodnotu $V_\tau(x_i)$.

i	1	2	3	4
$V_\tau(x_i)$	1	2	1	2

Tabulka 10: Vypočtené hodnoty V_τ

Z Tabulky 10 a vzorce 29 ze strany 45 můžeme sestavit matici $\mathbb{V}'$. Hodnoty z matice $\mathbb{V}'$ můžeme využít ke kontrole stupňů u vrcholů partity W grafu G. Sestavená matice $\mathbb{V}'$ je zobrazena v Tabulce 11.

$$\mathbb{V}' = \begin{array}{c} \\ \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \end{array} \begin{array}{c} \begin{array}{cccc} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} \end{array} \\ \begin{pmatrix} 1 & 2 & 1 & 2 \\ 2 & 4 & 2 & 4 \\ 1 & 2 & 1 & 2 \\ 2 & 4 & 2 & 4 \end{pmatrix} \end{array}$$

Tabulka 11: Matice počtu zápisu $\mathbb{V}'$ jednotlivým buněk v matici $\mathbb{V}$

Z Tabulky 11 je vidět, že bude potřeba použit právě čtyři disjunktní množiny na paralelní sestavení matice $\mathbb{V}$. Důvodem je to, že všechny čtyři lokálních matice $\mathbb{V}^{k,l}$ přistupují ke společným buňkám matice $\mathbb{V}$, tj. $\mathbb{V}[2,2]$, $\mathbb{V}[2,4]$, $\mathbb{V}[4,2]$ a $\mathbb{V}[4,4]$. V téhle chvíli již máme vše nachystané k tomu, abychom zkonstruovali graf G. Vrcholy z partity U grafu G se stanou dvojice oblastí

grafu S, tj. $f_k f_l$. Každý vrchol grafu G z partity U symbolizuje sestavení lokální matice $\mathbb{V}_{loc}^{k,l}$. V partitě W bude 16 vrcholů $V[m,n]$, kde $m,n \in \{1,2,\ldots,4\}$, které symbolizují buňky v globální matici $\mathbb{V}$. Hrany mezi partitami grafu G jsou jednoznačně určeny na základě buněk v matici $\mathbb{V}$, kde lokální matice $\mathbb{V}_{loc}^{k,l}$ přispívají svými příspěvky. Na Obrázku 15(a) je zobrazen vytvořený graf G. Následně pomocí grafu G vytvoříme graf H a to tak, že v grafu H spojíme s vrcholy

(a) graf G (b) graf H

Obrázek 15: Příklad grafu G a H pro jednoduchou síť

pouze tehdy, pokud mají společný vrchol v partitě W grafu G. Na Obrázku 15(b) vidíme, že nám vznikl graf H. Navíc graf H je současně i kompletním grafem K_4. Pokud bychom dobře barvili graf K_4, zjistíme, že na jeho dobré obarvení je potřeba použít právě čtyři barvy, viz Obrázek 15(b). Pro úplnost uvedeme matici $\mathbb{V}$, viz Tabulka 12. Matici $\mathbb{V}$ můžeme sestavit z grafu G nebo z lokálních matic $\mathbb{V}_{loc}^{k,l}$. Můžeme zkontrolovat, že počty sčítanců v matici $\mathbb{V}$ odpovídají hodnotám v matici $\mathbb{V}'$ z Tabulky 11.

$$\mathbb{V} = \begin{array}{c} \\ \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \end{array} \begin{array}{c} \begin{array}{cccc} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} \end{array} \\ \begin{pmatrix} \mathbb{V}^{1,1}[1,1] & \sum_{l=1}^{2} \mathbb{V}^{1,l}[1,2] & \mathbb{V}^{1,2}[1,3] & \sum_{l=1}^{2} \mathbb{V}^{1,l}[1,4] \\ \sum_{k=1}^{2} \mathbb{V}^{k,1}[2,1] & \sum_{k=1}^{2}\sum_{l=1}^{2} \mathbb{V}^{k,l}[2,2] & \sum_{k=1}^{2} \mathbb{V}^{k,2}[2,3] & \sum_{k=1}^{2}\sum_{l=1}^{2} \mathbb{V}^{k,l}[2,4] \\ \mathbb{V}^{2,1}[3,1] & \sum_{l=1}^{2} \mathbb{V}^{2,l}[3,2] & \mathbb{V}^{2,2}[3,3] & \sum_{l=1}^{2} \mathbb{V}^{2,l}[3,4] \\ \sum_{k=1}^{2} \mathbb{V}^{k,1}[4,1] & \sum_{k=1}^{2}\sum_{l=1}^{2} \mathbb{V}^{k,l}[4,2] & \sum_{k=1}^{2} \mathbb{V}^{k,2}[4,3] & \sum_{k=1}^{2}\sum_{l=1}^{2} \mathbb{V}^{k,l}[4,4] \end{pmatrix} \end{array}$$

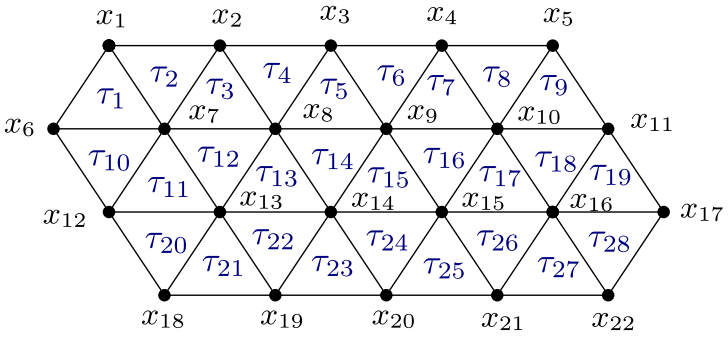
Tabulka 12: Matice $\mathbb{V}$

4.4.5 Konstrukce grafu H

Jak jsme mohli vidět z Příkladu 12, samotné grafy G a H jsou husté co do počtu hran. Algoritmicky konstruovat grafy G a H podle definice může být časově a hlavně paměťově náročné.

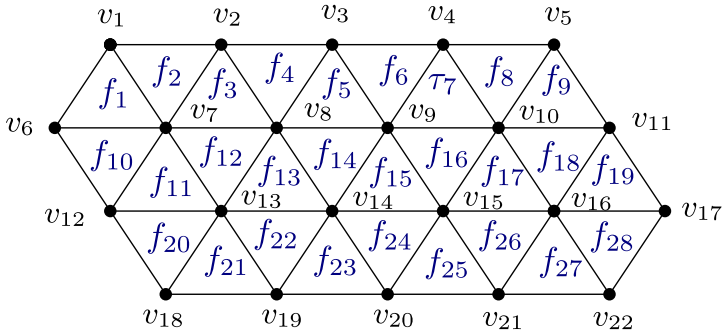
Místo toho se zaměříme, jak zkonstruovat pro každý vrchol grafu H množinu sousedních vrcholů, kterou můžeme využít v dobrém barvení grafu H při kontrole nepoužité barvy na okolí právě barveného vrcholu. Během barvení není potřeba znát celou strukturu grafu H, abychom dobře obarvili graf H.

Zaměřme se na jisté pozorování během sestavení lokálních matic $\mathbb{V}_{loc}^{k,l}$. Uvažujme nyní síť z Obrázku 16. Uzly sítě jsou označeny symbolem x_i, kde $i \in \{1, 2, \ldots, 22\}$. Elementy sítě (trojúhelníky) jsou označeny symbolem τ_j, kde $j \in \{1, 2, \ldots, 28\}$.



Obrázek 16: Síť

Pro úplnost uvádíme sestavený graf S v Obrázku 17. Vrcholy grafu S jsou označené symbolem v_i a oblasti grafu S jsou označeny symboly f_j.

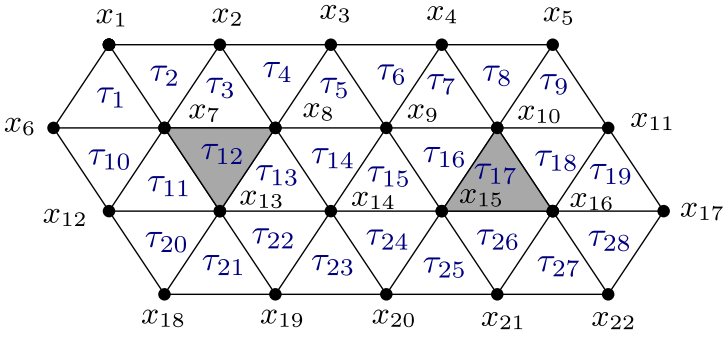


Obrázek 17: Graf S

Pro další fázi si vyberme jednu reprezentační dvojici elementů, např. (τ_{12}, τ_{17}), viz Obrázek 18. K této uspořádané dvojici elementů (τ_{12}, τ_{17}) sestavíme lokální matici $\mathbb{V}_{loc}^{12,17}$, viz Tabulka 13.

$$\mathbb{V}_{loc}^{12,17} = \begin{pmatrix} \mathbb{V}^{12,17}[7,10] & \mathbb{V}^{12,17}[7,15] & \mathbb{V}^{12,17}[7,16] \\ \mathbb{V}^{12,17}[8,10] & \mathbb{V}^{12,17}[8,15] & \mathbb{V}^{12,17}[8,16] \\ \mathbb{V}^{12,17}[13,10] & \mathbb{V}^{12,17}[13,15] & \mathbb{V}^{12,17}[13,16] \end{pmatrix}$$

Tabulka 13: Lokální matice souřadnic $\mathbb{V}_{loc}^{12,17}$



Obrázek 18: Síť s vybranou dvojicí oblastí (τ_{13}, τ_{17})

V tomto rozboru se zaměříme na dvojice elementů, jejíž lokální matice je v kolizi s maticí $\mathbb{V}_{loc}^{12,17}$ během zápisu svých hodnot do buněk matice $\mathbb{V}$. Zaměřme se podrobněji na buňky v lokální matici $\mathbb{V}_{loc}^{12,17}$, zde nás budou zajímat souřadnice buněk matice $\mathbb{V}$, do kterých se zapíše hodnota z lokální matice $\mathbb{V}_{loc}^{12,17}$. Symbolem $\times$ budeme značit operaci kartézského součinu. Uveďme si pro každou buňku matice $\mathbb{V}$ množinu dvojic elementů, jejíž lokální matice přičítá svůj příspěvek do této buňky:

$$\begin{aligned}
\mathbb{V}[7,10] &\ \ldots\ \{\tau_1,\tau_2,\tau_3,\tau_{10},\tau_{11},\tau_{12}\} \times \{\tau_7,\tau_8,\tau_9,\tau_{16},\tau_{17},\tau_{18}\},\\
\mathbb{V}[7,15] &\ \ldots\ \{\tau_1,\tau_2,\tau_3,\tau_{10},\tau_{11},\tau_{12}\} \times \{\tau_{15},\tau_{16},\tau_{17},\tau_{24},\tau_{25},\tau_{26}\},\\
\mathbb{V}[7,16] &\ \ldots\ \{\tau_1,\tau_2,\tau_3,\tau_{10},\tau_{11},\tau_{12}\} \times \{\tau_{17},\tau_{18},\tau_{19},\tau_{26},\tau_{27},\tau_{28}\},\\
\mathbb{V}[8,10] &\ \ldots\ \{\tau_3,\tau_4,\tau_5,\tau_{12},\tau_{13},\tau_{14}\} \times \{\tau_7,\tau_8,\tau_9,\tau_{16},\tau_{17},\tau_{18}\},\\
\mathbb{V}[8,15] &\ \ldots\ \{\tau_3,\tau_4,\tau_5,\tau_{12},\tau_{13},\tau_{14}\} \times \{\tau_{15},\tau_{16},\tau_{17},\tau_{24},\tau_{25},\tau_{26}\},\\
\mathbb{V}[8,16] &\ \ldots\ \{\tau_3,\tau_4,\tau_5,\tau_{12},\tau_{13},\tau_{14}\} \times \{\tau_{17},\tau_{18},\tau_{19},\tau_{26},\tau_{27},\tau_{28}\},\\
\mathbb{V}[13,10] &\ \ldots\ \{\tau_{11},\tau_{12},\tau_{13},\tau_{20},\tau_{21},\tau_{22}\} \times \{\tau_7,\tau_8,\tau_9,\tau_{16},\tau_{17},\tau_{18}\},\\
\mathbb{V}[13,15] &\ \ldots\ \{\tau_{11},\tau_{12},\tau_{13},\tau_{20},\tau_{21},\tau_{22}\} \times \{\tau_{15},\tau_{16},\tau_{17},\tau_{24},\tau_{25},\tau_{26}\},\\
\mathbb{V}[13,16] &\ \ldots\ \{\tau_{11},\tau_{12},\tau_{13},\tau_{20},\tau_{21},\tau_{22}\} \times \{\tau_{17},\tau_{18},\tau_{19},\tau_{26},\tau_{27},\tau_{28}\}.
\end{aligned}$$

Pro každou souřadnici $\mathbb{V}[j,i]$ (v našem případě) je potřeba znát šest elementů pro uzel x_j a šest elementů pro uzel x_i, ve kterých je daný uzel součástí. Množinu elementů, ve kterých je uzel x_j součástí si označme τ_{x_j} a analogicky množinu oblastí, ve kterých je vrchol x_i součástí si označme τ_{x_i}. Kartézský součin množin τ_{x_j} a τ_{x_i} nám udává, jaké dvojice elementů obsahují ve své lokální matici $\mathbb{V}_{loc}^{k,l}$ příspěvek, který zapisuje do buňky $\mathbb{V}[j,i]$.

Stejný princip lze popsat pomocí teorie grafu a to následovně. K dané sítí vytvoříme graf S. Připomeňme, že z uzlů sítě se stanou vrcholy grafu S a z elementů sítě se stanou oblasti grafu S. Množinu oblastí, ve kterých je vrchol v_i součástí, si označme F_{v_i} a obdobně množinu oblastí ve kterých je vrchol v_j součástí, si označme F_{v_j}. Množiny F_{v_i} a F_{v_j} jsou analogii zavedených množin τ_{x_i} a τ_{x_j}. V případě, že bychom chtěli dobře barvit graf H a byli ve fázi kontroly barvy u vrcholu

$f_{12}f_{17}$, zda nově přiřazená barva nekoliduje s ostatními vrcholy na okolí vrcholu $f_{12}f_{17}$. Bylo by potřeba sestavit množiny F_{v_7}, F_{v_8} a $F_{v_{13}}$ pro oblast f_{12} a množiny $Fv_{10}, F_{v_{15}}$ a $F_{v_{16}}$ pro oblast f_{17}. Pak bychom museli provést devět kartézských součinů množin $F_{v_o} \times F_{v_p}$, kde $o \in \{7, 8, 13\}$ a $p \in \{10, 15, 16\}$, ze kterých by vznikly dvojice oblastí, tj. vrcholy grafů H, které nesmí být obarvené stejnou barvou jako vrchol (f_{12}, f_{17}). Znamenalo by to pro každý vrchol grafu H provést alespoň $9 \cdot 36 = 324$ testů, zda nově přiřazená barva není v kolizi. Číslice devět vzešla z vytvoření devíti kartézských součinů (stejně jako je řád lokální matice $\mathbb{V}^{k,l}_{loc}$) a číslo 36 vzniklo z kartézského součinu, kde spolu vždy interagovaly šesti prvkové množiny $F_{v_7}, F_{v_8}, F_{v_{13}}, F_{v_{10}}, F_{v_{15}}$ a $F_{v_{16}}$. Některé vrcholy, jak brzy zjistíme, jsou kontrolovány zbytečně vícekrát.

Zaměřme se na redukci počtu duplicitních kontrol během dobrého barvení grafu H. Vrcholy trojúhelníkové oblasti f_k si označme v_{k1}, v_{k2} a v_{k3}, obdobně vrcholy trojúhelníkové oblasti f_l si označme v_{l1}, v_{l2} a v_{l3}. Sestavme lokální matici $\mathbb{V}^{k,l}_{loc}$ pro oblast $f_k f_l$, viz Tabulka 14.

$$\mathbb{V}^{k,l}_{loc} = \begin{pmatrix} \mathbb{V}^{k,l}[k1, l1] & \mathbb{V}^{k,l}[k1, l2] & \mathbb{V}^{k,l}[k1, l3] \\ \mathbb{V}^{k,l}[k2, l1] & \mathbb{V}^{k,l}[k2, l2] & \mathbb{V}^{k,l}[k2, l3] \\ \mathbb{V}^{k,l}[k3, l1] & \mathbb{V}^{k,l}[k3, l2] & \mathbb{V}^{k,l}[k3, l3] \end{pmatrix}$$

Tabulka 14: Lokální matice souřadnic $\mathbb{V}^{k,l}_{loc}$

Předpokládejme, že každý z vrcholů $v_{k1}, v_{k2}, v_{k3}, v_{l1}, v_{l2}$ a v_{l3} je součástí právě šesti oblastí. Označme si šest šesti prvkových množin oblastí, tj. $F_{v_{k1}}, F_{v_{k2}}, F_{v_{k3}}, F_{v_{l1}}, F_{v_{l2}}$ a $F_{v_{l3}}$, kde jsou konkretní vrcholy $v_{k1}, v_{k2}, v_{k3}, v_{l1}, v_{l2}$ a v_{l3} součástí těchto oblastí. Pro každou souřadnici matice $\mathbb{V}$, do které se přičítá hodnota z lokální matice $\mathbb{V}^{k,l}_{loc}$, uveďme množinu dvojic oblastí, jejíž sestavená lokální matice zapisuje do stejné buňky v matici $\mathbb{V}$.

$$\begin{aligned}
&\mathbb{V}[k1, l1] \ldots F_{v_{k1}} \times F_{v_{l1}}, \\
&\mathbb{V}[k1, l2] \ldots F_{v_{k1}} \times F_{v_{l2}}, \\
&\mathbb{V}[k1, l3] \ldots F_{v_{k1}} \times F_{v_{l3}}, \\
&\mathbb{V}[k2, l1] \ldots F_{v_{k2}} \times F_{v_{l1}}, \\
&\mathbb{V}[k2, l2] \ldots F_{v_{k2}} \times F_{v_{l2}}, \\
&\mathbb{V}[k2, l3] \ldots F_{v_{k2}} \times F_{v_{l3}}, \\
&\mathbb{V}[k3, l1] \ldots F_{v_{k3}} \times F_{v_{l1}}, \\
&\mathbb{V}[k3, l2] \ldots F_{v_{k3}} \times F_{v_{l2}}, \\
&\mathbb{V}[k3, l3] \ldots F_{v_{k3}} \times F_{v_{l3}}.
\end{aligned}$$

Nyní sjednotíme jednotlivé množiny dvojic oblastí, jejíž lokální matice koliduje při zápisu s lokální maticí $\mathbb{V}_{loc}^{k,l}$.

$$\begin{aligned}
&\bigcup_{\substack{k\in\{k1,k2,k3\},\\ l\in\{l1,l2,l3\}}} F_{v_k} \times F_{v_l} = (F_{v_{k1}} \times F_{v_{l1}}) \cup (F_{v_{k1}} \times F_{v_{l2}}) \cup (F_{v_{k1}} \times F_{v_{l3}}) \cup (F_{v_{k2}} \times F_{v_{l1}}) \cup \dots \\
&\dots \cup (F_{v_{k2}} \times F_{v_{l2}}) \cup (F_{v_{k2}} \times F_{v_{l3}}) \cup (F_{v_{k3}} \times F_{v_{l1}}) \cup (F_{v_{k3}} \times F_{v_{l1}}) \cup (F_{v_{k3}} \times F_{v_{l3}}) = \dots \\
&\dots = [F_{v_{k1}} \times (F_{v_{l1}} \cup F_{v_{l2}} \cup F_{v_{l3}})] \cup [F_{v_{k2}} \times (F_{v_{l1}} \cup F_{v_{l2}} \cup F_{v_{l3}})] \cup [F_{v_{k3}} \times (F_{v_{l1}} \cup F_{v_{l2}} \cup F_{v_{l3}})] = \dots \\
&\dots = (F_{v_{k1}} \cup F_{v_{k2}} \cup F_{v_{k1}}) \times (F_{v_{l1}} \cup F_{v_{l2}} \cup F_{v_{l3}})
\end{aligned} \tag{31}$$

Ze vztahu 31 vidíme, že k jednotlivým oblastem f_k a f_l si stačí uchovávat množinu všech oblastí, se kterými daná oblast sousedí přes libovolný vrchol včetně oblasti samotné. Stejný pohled platí i pro uzly a elementy sítě. Vraťme se k našemu příkladu s dvojicí oblastí (f_{12}, f_{17}), kterou bychom chtěli dobře obarvit v grafu H. K dobrému obarvení vrcholu (f_{12}, f_{17}) v grafu H je potřeba sestrojit množinu sousedních vrcholů $N_H(f_{12}, f_{17})$. Ke splnění podmínky dobrého vrcholového barvení nesmí být přiřazená barva vrcholu v_i použita u vrcholu z množiny $N(v_i)$. V našem případě to znamená, že barvu, kterou barvící algoritmus přiřadí vrcholu (f_{12}, f_{17}) už nesmí být obarven žádný vrchol z množiny $N_H(f_{12}, f_{17})$. Pokud by existoval vrchol z množiny $N_H(f_{12}, f_{17})$ takový, že by měl přiřazenou stejnou barvu jako vrchol (f_{12}, f_{17}), znamenalo by to kolizi při sestavení matice $\mathbb{V}$. K sestavení množiny $N_H(f_{12}, f_{17})$ využijeme Vztah 31. K tomu, abychom mohli použit Vztah 31 je nutné znát ke každé oblasti grafu S množinu sousedních oblastí. V našem případě to znamená uchovávat si k oblasti f_{12} množinu sousedních oblastí $F_{v_7} \cup F_{v_8} \cup F_{v_{13}}$ a analogicky pro oblast f_{17} si budeme uchovávat množinu sousedních oblastí $F_{v_{10}} \cup F_{v_{15}} \cup F_{v_{16}}$.

Pokud nyní využijeme Vztah 31, do kterého dosadíme za množinu $\{k1, k2, k3\}$ množinu $\{7, 8, 13\}$ a za množinu $\{l1, l2, l3\}$ množinu $\{10, 15, 16\}$, obdržíme množinu dvojic oblastí, tj. vrcholy grafu H, jejíž lokální matice $V_{loc}^{k,l}$ koliduje se zápisem do matice $\mathbb{V}$ s lokální matici $V_{loc}^{12,17}$ z Tabulky 13. Je důležité si uvědomit, že ve vzniklé množině dvojic oblastí je i dvojice (f_{12}, f_{17}). Proto když budeme sestavovat množinu $N_H(f_{12}, f_{17})$ pomocí rovnice 31 je nutné množinově odečíst samotnou dvojici (f_{12}, f_{17}). Pro množinu $N_H(f_{12}f_{17})$ obdržíme předpis

$$\begin{aligned}
N_H(f_{12}f_{17}) &= ((F_{v_7} \cup F_{v_8} \cup F_{v_{13}}) \times (F_{v_{10}} \cup F_{v_{15}} \cup F_{v_{16}})) \setminus \{f_{12}f_{17}\}, \\
N_H(f_{12}f_{17}) &= (\{f_1, f_2, f_3, f_4, f_5 f_{10}, f_{11}, f_{12}, f_{13}, f_{14}, f_{20}, f_{21}, f_{22}\} \times \{f_7, f_8, f_9, \dots \\
&\qquad \dots, f_{15}, f_{16}, f_{17}, f_{18}, f_{19}, f_{24}, f_{25}, f_{26}, f_{27}, f_{28}\}) \setminus \{f_{12}f_{17}\}.
\end{aligned} \tag{32}$$

Definujme množinovou funkci FN z anglického spojení slov *Face Neighbors*, která nám pro zadanou oblast grafu S vrátí množinu oblastí, se kterými je daná oblast sousední přes vrchol.

Definice 29 *Mějme rovinný graf $S = (V, E)$. Množinu oblastí grafu S označme F_S. Operaci $FN : F_S \to 2^{F_S}$ nazvěme množinovou funkcí Face Neighbors, pokud pro $\forall f_i \in F_S$ množinová funkce FN vrátí množinu všech oblastí, se kterými má společný alespoň jeden vrchol v grafu S.*

V Definici 29 připouštíme situaci, kdy pro každou oblast f grafu S platí, že je obsažena v množině oblastí $FN(f)$. Obecný vztah pro výpočet množiny sousedních vrcholů grafu H pro trojúhelníkovou síť potom vypadá následovně.

$$\begin{aligned} &\forall (f_i, f_j) \in V(f_i, f_j) : N_H(f_i, f_j) = (FN(f_i) \times FN(f_j)) \setminus \{(f_i, f_j)\}, \\ &\forall (f_i, f_j) \in V(f_i, f_j) : N_H[(f_i, f_j)] = FN(f_i) \times FN(f_j) \end{aligned} \tag{33}$$

Když známe množinu sousedních vrcholů, umíme snadno spočítat stupeň vrcholů v grafu H. Stupeň vrcholu (f_{12}, f_{17}) v grafu H je $|N_H(f_{12}f_{17})| = 13^2 - 1 = 168$. Stupeň vrcholu v grafu H nám udává počet kontrol, které budeme muset provést pro každý vrchol během přiřazení nové barvy.

Pokud budeme mít trojúhelníkovou síť, kde každý uzel má na svém okolí šest elementů, pak bude nutné pro každý vrchol v grafu H provést 168 kontrol. Můžeme uvést Lemma 5, které udává obecný vzorec pro výpočet sousedních oblastí, které jsou vrcholově sousední k zadané oblasti f v rovinném grafu G, který tvoří pouze trojúhelníkové oblasti. Důkaz Lemmatu 5 je dílem autora této práce.

Lemma 5 *Mějme rovinný souvislý graf $G = (V, E)$, kde každá oblast je trojúhelník, $\delta(G) \geq 3$ a navíc každý vrchol v grafu G je součástí právě* $\deg(v)$ *oblastí, pak pro každou oblast $f \in F$ platí, že je sousední přes vrchol s*

$$\sum_{v \in V(f)} \deg(v) - 5 \text{ } \textit{oblastmi včetně oblasti samotné.}$$

Označení $V(f)$ v Lemmatu 5 rozumíme množinu vrcholů, která tvoří trojúhelníkovou oblast f.

Důkaz Zvolme si libovolnou oblast f rovinného grafu G. Vrcholy trojúhelníkové oblasti f si označme v_1, v_2 a v_3 a hrany oblasti f si označme $e_1 = \{v_1, v_2\}$, $e_2 = \{v_2, v_3\}$ a $e_3 = \{v_3, v_1\}$. Součet stupňů vrcholů, které tvoří oblast f si označme $y := \sum_{v \in \{v_1, v_2, v_3\}} \deg(v)$. Předpokládáme, že každý vrchol je součástí právě $\deg(v)$ oblastí, proto samotná oblast f je v y započítaná třikrát za každý vrchol, proto od y odečteme dva duplicitní výskyty oblasti f. Dále na každé hraně oblastí f leží právě jedna oblast f_1. Bez újmy na obecnosti předpokládejme, že hrana $e_1 = \{v_1, v_2\}$ je hranou oblasti f_1. Protože $\delta(G)$ je alespoň tři, musí proto v grafu G mít každý vrchol stupeň alespoň 3. Proto v grafu G existuje další vrchol v_4, který je různý od vrcholů v_1,v_2 a v_3 a navíc je spojený s vrcholy v_1 a v_2 (předpokládáme pouze trojúhelníkové oblasti), jinak by stupeň vrcholů v_1, v_2, v_3 byl dva. Jednalo by se o graf C_3, který nesplňuje podmínku Lemmatu 5

$\delta(G) \geq 3$. Nyní víme, že trojúhelníkovou oblast f_1 tvoří vrcholy v_1, v_2, v_4. A při sčítání stupňů vrcholů v y jsme oblast f_1 započítali dvakrát, jednou za vrchol v_1 kvůli hraně $e_4 = \{v_1, v_4\}$ a po druhé za vrchol v_2 kvůli hraně $\{v_2, v_4\}$. Proto od y odečteme jedničku za každou hranu oblasti f. Obdržíme pak hodnotu $y - 2 - 3 = y - 5$, což je informace o počtech oblastí, které jsou sousední s oblastí f přes vrcholy včetně oblastí samotné. ∎

Není těžké si uvědomit, že vzorec, který je uveden v Lemmatu 5 nám dává informaci o počtech prvků v množině $FN(f)$, ale pouze pro rovinné grafy, kde každá oblast f grafu S je trojúhelník. Ověřme, zda pomocí vzorce z Lemmatu 5 obdržíme stejný výsledek pro určení stupně vrcholu $f_{12}f_{17}$ v grafu H. Nejdříve spočtěme počet sousedních oblastí k oblasti f_{12} grafu S. Víme, že stupeň vrcholů v_7, v_8 a v_{13} je 6, viz Obrázek 17. Při použití vzorce z Lemmatu 5 obdržíme počet sousedních oblastí k oblasti f_{12}, kterých je

$$\sum_{v \in \{v_7, v_8, v_{13}\}} \deg(v) - 5 = 6 + 6 + 6 - 5 = 13.$$

Stejný výsledek obdržíme i pro oblast f_{17} grafu S, tj. počet sousedních oblastí k oblasti f_{17} je

$$\sum_{v \in \{v_{10}, v_{15}, v_{16}\}} \deg(v) - 5 = 6 + 6 + 6 - 5 = 13.$$

Víme, že vrcholy v grafu H jsou tvořeny dvojicemi oblastí grafu S, které jsou sousední přes vrcholy bez oblasti samotné. Snadno tak spočteme stupeň vrcholu $f_{12}f_{17}$ grafu H jako

$$\deg_H(f_{12}f_{17}) = \left(\sum_{v \in \{v_7, v_8, v_{13}\}} \deg(v) - 5 \right) \cdot \left(\sum_{v \in \{v_{10}, v_{15}, v_{16}\}} \deg(v) - 5 \right) - 1 = 13 \cdot 13 - 1 = 168.$$

Vidíme, že stupeň vrcholu nám vyšel stejně jako velikosti množiny $|N_H(f_{12}f_{17})|$. Můžeme uvést obecný vztah pro stupeň vrcholů grafu H, který vypadá následovně

$$\forall f_j f_i \in V(H): \quad \deg_H(f_j f_i): \left(\sum_{v \in V(f_j)} \deg(v) - 5 \right) \cdot \left(\sum_{v \in V(f_i)} \deg(v) - 5 \right) - 1, \qquad (34)$$

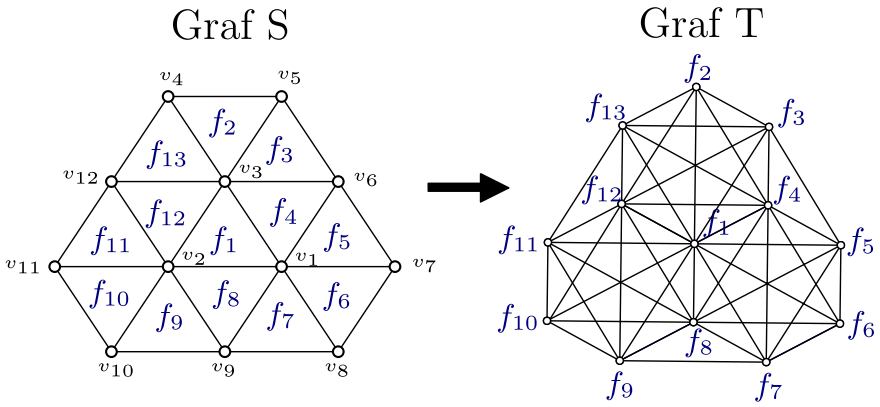
kde f_i, f_j jsou trojúhelníkové oblasti rovinného grafu S. Pro další rozbor úlohy předpokládejme, že síť je složena pouze z trojúhelníků a maximální počet oblastí, které se potkávají v jednom uzlu sítě je šest. Z toho vyplývá, že maximální stupeň grafu H je 168. Sestavení množiny sousedních vrcholů pro graf H využijeme v prvním algoritmu barvení grafu H.

4.4.6 Popis grafu T

V této sekci popíšeme nový graf T, následně dáme tento nový graf T do vztahu s grafem H. Uveďme nejprve definici nového grafu T.

Definice 30 *Nechť máme graf $S = (V, E)$ a F je množina oblastí grafu S, potom graf $T = (F, E')$ vznikne tak, že oblasti grafu S prohlásíme za vrcholy grafu T. Množina hran grafu T bude $E' = \{f_i f_j\}$ pro $f_i, f_j \in F$, kde $f_i \neq f_j$. Hrana v grafu T, bude existovat mezi vrcholy, právě tehdy, když oblasti f_i a f_j v grafu S sdílejí alespoň jeden společný vrchol. Tento postup výroby grafu T z rovinného grafu S, budeme nadále v tomto textu značit $\eta(S) \to T$.*

Příklad 13
Pro lepší pochopení uvádíme příklad grafu S a k němu vytvořený graf T, viz Obrázek 19. Graf T není duální graf ke grafu S ve smyslu duálního grafu z teorie grafů. Sousednost v grafu T určujeme přes vrcholy v grafu S a né přes hrany, tak jak je tomu v duálním grafu rovinného grafu.



Obrázek 19: Sestavení grafu T

Poznamenejme, že okolí každého vrcholu v grafu T můžeme zkonstruovat pomocí množinové funkce FN, viz Definice 29 ze strany 55 tímto způsobem

$$\begin{aligned} &\forall f \in F_S : \ N_T(f) = FN(S) \setminus \{f\}, \\ &\forall f \in F_s : \ \ N_T[f] = FN(S). \end{aligned} \tag{35}$$

Podíváme-li se na vzorec 33 ze strany 56 vidíme, že výpočet $N_H[(f_i, f_j)]$ můžeme odpovídá vzorci 2 ze strany 29, kdy popisujeme množinu sousedních vrcholů grafu, který vznikl silným součinem. Protože víme, že každé okolí se dá sestavit pomocí vzorce 33, můžeme dospět k tomu, že graf H je silným součinem dvou grafů T

$$\forall (f_i, f_j) \in V(H) : N_H[(f_i, f_j)] = FN(f_i) \times FN(f_j) = N_T[f_i] \times N_T[f_j] = N_{T \boxtimes T}[(f_i, f_j)]. \tag{36}$$

Množinu hran grafu H bychom mohli popsat

$$E(H) = \{(f_i, f_j)(f_k, f_l) : (i = k \wedge f_j f_l \in E(T)) \vee (j = l \wedge f_i f_k \in E(T)) \vee (f_i f_k, f_j f_l \in E(T))\}, \tag{37}$$

kde $i, j, k, l \in \{1, 2, \ldots, |V(T)| = F\}$. Pokud by v 37 nebyla podmínka na hrany $(f_i f_k, f_j f_l \in E(T))$, jednalo by se o hranovou množinu grafu $E(T \square T)$. Pokud by ovšem ve vzorci byla pouze podmínka $(f_i f_k, f_j f_l \in E(T))$, pak by se jednalo o hranovou množinu $E(T \times T)$. Sjednocení těchto dvou množin získáváme silný součin dvou grafů T, tj. graf H, viz Definice 17, která se nachází na straně 28. Pro uzavřené okolí N_H a $N_{T \boxtimes T}$ každého vrcholu v grafu H, a množina hran je totožná, navíc víme, že počet vrcholů grafu H a $T \boxtimes T$ je v obou případech F^2, jedná se tak o stejný graf a proto platí $H = T \boxtimes T$.

Pro naši zvolenou síť a k ní vytvořený graf S platí, že maximální stupeň vrcholu v grafu S je 6. Předpokládejme, že v grafu S existuje trojúhelníková oblast f^*, kde každý vrchol má stupeň 6. Pokud použijeme Lemma 5 ze strany 56 při výpočtu sousedních oblastí k oblasti f^*, obdržíme hodnotu 13, ve které je obsažená i oblast f^*. Pokud od 13 odečteme 1 za samotnou oblast f^*, obdržíme maximální stupeň v grafu T, tj. $N_T(f^*) = \Delta(T) = 12$. Podle Lemmatu 2 ze strany 22, platí, že horní odhad chromatického čísla v libovolného grafu G je dán vzorcem $\Delta(G) + 1$, tj. $\chi(G) < \Delta(G) + 1$. Pokud za graf G vezmeme graf T získáme horní odhad počtu barev grafu T, který je 13. S využitím Věty 4 ze strany 29 platí, že chromatické číslo grafu H je

$$\chi(H) = \chi(T \times T) \leq \chi(T)\chi(T) \leq (\Delta(T) + 1)^2 = (12 + 1)^2 = 169. \tag{38}$$

4.5 Shrnutí

V této kapitole jsme se nejprve zabývali problémem vzniku kolizí. Dále jsme se uvedli sestavení matice $\mathbb{V}'$, která nám reprezentuje jednotlivé počty přístupů do buněk v globální matici $\mathbb{V}$. V další podkapitole jsme se zabývali grafovým popisem tří grafů S, G a H. Kde pomocí grafu H popisujeme kolize při zápisu hodnot z lokálních matic do buněk v globální matici $\mathbb{V}$. Nastínili jsme případné řešení problému s kolizemi, založené na dobrém barvení grafu H. V poslední podkapitole jsme se zabývali efektivním sestavením grafu H pomocí určení množiny sousedních vrcholů k danému vrcholu v grafu H.

5 Algoritmy dobrého vrcholového barvení grafu H

V této kapitole popíšeme tři algoritmy dobrého barvení grafu H, které budou založeny na heuristických metodách. Ačkoliv by se nám líbilo obarvit graf H pomocí $\chi(H)$ barev, deterministické algoritmy mají exponenciální složitost. Proto jsme se museli uchýlit k heuristickým barvícím algoritmům, kde je jejich rychlost kompenzována obarvením grafu H pomocí více barev než je $\chi(H)$. Předpokládejme, že vstupní síť tvoří pouze z trojúhelníkové elementy a navíc každý uzel bude vrcholem nejvýše šesti elementů. Sítě s takovými vlastnosti vznikají během zjemňování sítě, viz Obrázek 13 ze strany 49. Barvy v dobrém barvení grafu H budeme reprezentovat přirozenými čísly s číslem 0. Nulová barva bude mít speciální význam. První algoritmus, který uvedeme, bude využívat hladové barvení grafu H. Připomeňme, že hladový algoritmus přiřadí každému vrcholu nejnižší číselnou hodnotu barvy na jeho okolí.

5.1 Barvení grafu H pomocí hladového algoritmu

Sekvenční Algoritmus 7 rozdělíme na tři části, které postupně popíšeme v jednotlivých sekcích. V první sekci sestavíme pole nejvýše šesti prvkových množin *NodeElements*. V tomto poli budeme pro každý uzel sítě uchovávat elementy, ve kterých je konkrétní uzel obsažen. V druhé části budeme sestavovat až třinácti prvkové množiny elementů pro každý element sítě, které uložíme do pole množin *ElementNeighbours*. Do pole množin *ElementNeighbours* budeme ukládat sousední elementy k zadanému elementu. Slovem sousední rozumíme v tomto pojetí sousední přes uzel sítě. V třetí částí se zaměříme na samotné barvení grafu H. Vrcholy grafu H, tj. dvojice oblastí grafu S, budou nyní dvojice elementů sítě. Na dobré obarvení grafu H využijeme hladový Algoritmus 3 ze strany 26, kvůli své malé časové náročnosti. Samotná rychlost barvícího algoritmu je klíčová, aby samotné barvení netrvalo déle než sestavení matice $\mathbb{V}$. Připomeňme, že Algoritmus 3 nezaručuje nalezení optimálního počtu barev. Pro vrcholy grafu H, které jsou obarvené stejnou barvou platí, že je možné zápisy hodnot z jejich sestavených lokálních matic do buněk matice $\mathbb{V}$, provádět paralelně s jistotou, že nenastane kolize.

5.1.1 Vstup Algoritmu 7

Vstupem Algoritmu 7 bude množina všech elementů, kterou budeme značit $\{\tau\}$. i-tý prvek z množiny $\{\tau\}$ budeme značit τ_i, tj. i-tý element sítě. Dále symbolem N budeme značit počet uzlů sítě a symbolem F budeme značit počet elementů sítě.

5.1.2 Sestavení pole množin *NodeElements*

V této sekci se budeme zabývat 1. až 6. řádkem v Algoritmu 7. Naším cílem v této sekci je sestavit až šesti prvkové množiny elementů pro všechny uzly sítě. Každý uzel sítě x_i bude v množině $NodeElements[x_i]$ obsahovat pouze ty elementy, ve kterých je vrcholem, těch může být maximálně šest, kvůli volbě vstupní sítě. Symbolem x_j budeme značit j-tý uzel sítě a označením $x_j \in \tau_i$ budeme rozumět tak, že se jedná o vrchol elementu τ_i. Pole *NodeElements* vytvoříme tak, že budeme procházet každý elementy sítě a každému uzlu, který je vrcholem elementu, uložíme informaci o tom, že je vrcholem konkrétního elementu. Vnitřní smyčka se provede pro každý element sítě třikrát neboť máme pouze síť složenou z trojúhelníkových elementů.

Časová složitost sestavení pole *NodeElements* je $3F \in \mathcal{O}(F)$, kde F je počet elementů sítě. Paměťová složitost sestavení pole *NodeElements* je $6N \in \mathcal{O}(N)$, kde N je počet uzlů sítě.

5.1.3 Sestavení pole množin *ElementNeighbors*

V této sekci se zaměříme na 7. až 12. řádek v Algoritmu 7. Cílem Algoritmu 7 v této sekci je sestavit množinu sousedních elementů pro každý element sítě. Každý element sítě τ_j bude v množině $ElementNeighbors[\tau_j]$ obsahovat ty elementy, se kterými element τ_j sdílí alespoň jeden společný uzel v sítí. V každé množině $ElementNeighbours[\tau_j]$ bude uložen element τ_j. Z vlastností vstupní sítě vyplývá, že horní odhad počtu sousedních elementů k zadanému elementu je 13. Pole *ElementNeighbours* vytvoříme tak, že budeme procházet každý elementy sítě, ke kterému uložíme množinu sousedních vrcholů, které jsou s tímto elementem sousední přes libovolný uzel v síti. Ke zjištění oblastí, ve kterých je konkretní uzel součástí, využijeme vytvořené pole *NodeNeighborsElements* z předchozí sekce. Vnitřní *For* smyčka v Algoritmu 7 se provede vždy třikrát, protože máme trojúhelníkové elementy a tedy každá oblast naší sítě je tvořena třemi uzly.

Časová složitost sestavení pole *ElementNeighbours* je $3\Delta(S)F = 18F \in \mathcal{O}(F)$. Ke konstantě 18 jsme se dopočítali tak, že každá oblast má 3 uzly a pro každý uzel pole může mít množina $NodeElements[x_j]$ až 6 oblastí, které je potřeba sjednotit s množinou $NodeElements[\tau_j]$. V součtu tak obdržíme celkem 18 operací sjednocení pro každý element. Paměťová složitost je $13F \in \mathcal{O}(F)$, jedná se o horní odhad velikosti pole *ElementNeighbours*.

Na pole *ElementNeighbors* se můžeme dívat tak, že se jedná o analogii s množinovou funkcí FN ze strany 55. Kde vrcholy grafu S jsou v našem případě uzly sítě a oblasti grafu S jsou elementy sítě. Rovinným grafem S se pak stává síť, u které předpokládáme, že je možné ji popsat rovinným grafem.

5.1.4 Sestavení matice *ColorMatrix*

V této sekci se zaměříme na 13. až 25. řádek v Algoritmu 7. Cílem v této sekci je dobře obarvit graf H, kde vrcholy grafu H reprezentujeme pomocí dvojic elementů sítě. Protože se na předchozí pole *ElementNeighbours* můžeme dívat jako na množinovou funkci FN, pak lze využít vzorec 33 ze strany 56 k určení množiny sousedních vrcholů v grafu H. Ve dvourozměrné matici *ColorMatrix* budeme uchovávat celočíselné hodnoty barev, přiřazené dvojici elementů. Výchozí hodnota barev v matici *ColorMatrix* je 0. Nulová barva nám bude reprezentovat situaci, kdy daná dvojice elementů zatím nebyla obarvená. Matici *ColorMatrix* sestavíme tak, že budeme procházet všechny dvojice elementů, tj. vrcholy grafu H. Pro každou dvojici elementů sestavíme množinu sousedních dvojic elementů v grafu H pomocí vzorce 33, kde za množinovou funkci FN vezmeme pole *ElementNeighbours*. Pro každou dvojici elementů nastavíme na začátku barvení všem buňkám pole *usedColors* hodnotu *False* pomocí metody *setAllCellsToFalse*. Hodnota *False* v poli *usedColors* má význam zatím nepoužité barvy na okolí grafu H. Potom pro každý vrchol grafu H, tj. dvojice elementů (τ_i, τ_j), procházíme množinu sousedních vrcholů, tj. dvojice elementů $\tau_k\tau_l$, kde do pole *usedColor* zaznamenáváme hodnotu *True* použitým barvám na okolí právě barveného vrcholu. Po navštívení všech sousedních vrcholů v grafu H pro zadaný vrchol, zavoláme nad polem *usedColors* metodu *findFirstFalseIndex*. Metoda *findFirstFalseIndex* nám vrátí nejnižší nenulovou hodnotu nepoužité barvy v poli *usedColors*, kterou je možné aktuální vrchol, tj. dvojice elementů (τ_i, τ_j), obarvit. Na konci Algoritmu 7 máme v matici *ColorMatrix* dobře vrcholově obarvený graf H, který je tvořen dvojicemi elementů.

Víme, že horní odhad počtu barev v dobrém vrcholovém barvení je dán vztahem $\Delta(G) + 1$ pro obecný graf G, viz Lemma 2 ze strany 22. Pro náš graf H, který závisí na vlastnostech sítě, jsme v předchozí kapitole vypočetli maximální stupeň grafu H jako $\Delta(H) = 168$, viz vzorec 34 ze strany 57. Podle Lemmatu 2 dostáváme hodnotu horního odhadu 169 barev pro náš graf H. Protože v Algoritmu 7 pracujeme s virtuální nulovou barvou, je potřeba s touto barvou počítat při vytvoření pole *usedColor*. Při kontrole barvy okolních vrcholů kontrolujeme navíc i barvu právě barveného vrcholu. To naštěstí nevadí, neboť právě obarvovaný vrchol má v matici *ColorMatrix* nulovou barvu. Přidání podmínky, která by ošetřila tuto jednu kontrolu by znamenalo zhoršení času sestavení matice *ColorMatrix*.

Časová složitost barvení je $508F^2 \in \mathcal{O}(F^2)$. Ke konstantě 508 jsme dospěli tak, že maximální stupeň grafu H je $\Delta(H) = 168$. Během kontroly sousedních vrcholů pro každý vrchol grafu H však kontrolujeme navíc barvu samotného barveného vrcholu, což nám dává v součtu 169 kontrol. Při zavolání metody *setAllCellsToFalse* je potřeba všem 170 buňkám v poli *usedColor* přistoupit a nastavit hodnotu na *False*. Navíc při zavolání metody *findFirstFalseIndex* se v nejhorším případě nachází buňka s hodnotou *false* na posledním místě, což je 169 kontrol (nulovou barvu nekontrolujeme). Počet vrcholů grafu H je F^2. Paměťová náročnost obarvení grafu H je $\mathcal{O}(F^2)$,

neboť potřebujeme uložit barvu každé uspořádané dvojici elementů v matici *ColorMatrix*.

Algorithm 7 Dobré vrcholové barvení grafu H

Require: nodes count N, elements count F, Set of elements $\{\tau\}$

```
NodeElemtents ← ArraySet[N][6]
for all element τ_i ∈ {τ} do
    for all node x_j ∈ τ_i do
        NodeElemtents[x_i] ← NodeElemtents[x_j] ∪ {τ_j}
    end for
end for
ElementNeighbors ← ArraySet[F][13]
for all element τ_i ∈ {τ} do
    for all node x_j ∈ τ_i do
        ElementNeighbors[τ_i] ← ElementNeighbors[τ_i] ∪ NodeElemtents[x_j]
    end for
end for
ColorMatrix ← Zero2DArray[F][F]
usedColors ← BoolArray[170]
for all element τ_i ∈ {τ} do
    for all element τ_j ∈ {τ} do
        usedColors.setAllCellsToFalse()
        for all element τ_k ∈ ElementNeighbors[τ_i] do
            for all element τ_l ∈ ElementNeighbors[τ_j] do
                usedColors[ColorMatrix[τ_k][τ_l]] = True
            end for
        end for
        ColorMatrix[τ_i][τ_j] ← usedColors.findFirstFalseIndex()
    end for
end for
```

5.1.5 Časová a paměťová složitost Algoritmu 7

Celková časová složitost algoritmu je dána součtem časových složitostí v jednotlivých sekcích, tj. $3F + 18F + 170F^2 = 21F + 508F^2 \in \mathcal{O}(F^2)$. Největší časovou složitost má samotné barvení grafu H.

Celková paměťová složitost algoritmu je taktéž dána součtem paměťových složitostí jednotlivých sekcích, tj. $6N + 13F + F^2 = 16F - 12 + F^2 \in \mathcal{O}(F^2)$. V odhadu jsme využili Důsledek 1 ze strany 19, kde pro počet oblastí rovinného grafu platí, že je ekvivalentní dvojnásobku počtu hran bez čtyř.

5.1.6 Zpracování matice *ColorMatrix*

V Algoritmu 7 na straně 63 jsme ukázali postup jak lze dobře obarvit graf H. V matici *ColorMatrix* jsou uložené barvy k jednotlivým uspořádaných dvojicím elementům. Abychom samotné barvení mohli využít v algoritmu sestavení matice $\mathbb{V}$, je potřeba vytvořit barevné třídy grafu H, které budeme reprezentovat pomocí pole množin *ColorClass*. Indexem v poli *ColorClass* budeme rozumět hodnotu barvy v dobrém barvení grafu H a množině *ColorClass*[color] budeme rozumět tak, že se jedná o barevnou třídu grafu H barvy *color*. Algoritmus 8 musí projdít všechny uspořádané dvojice elementů v matici *ColorMatrix* a zařadit tyto dvojice elementů k příslušné barevné třídě.

Časová složitost Algoritmu 8 je $F^2 \in \mathcal{O}(F^2)$, neboť je potřeba projít všechny dvojice elementů, kterých je F^2. Paměťová složitost Algoritmu 8 je $2F^2 \in \mathcal{O}(F^2)$, neboť je třeba uložit všechny dvojice elementů do pole *ColorClass*.

Algorithm 8 Zpracování pole *ColorMatrix*

Require: ColorMatrix, set of elements $\{\tau\}$

```
1: ColorClass ← ArraySet[169]
2: for all element τ_i ∈ {τ} do
3:     for all element τ_j ∈ {τ} do
4:         currentColor ← ColorMatrix[τ_i][τ_j]
5:         ColorClass[currentColor] ← ColorClass[currentColor] ∪ {(τ_i, τ_j)}
6:     end for
7: end for
```

5.1.7 Shrnutí

V této sekci jsme ukázali funkční Algoritmus 7, který má časovou i paměťovou náročnost $\mathcal{O}(F^2)$, kde F je počet elementů sítě. Pokud bychom zpracovávali síť o velikosti 10^6 pak samotnému uložení matice *ColorMatrix* budeme potřebovat 4 TB paměti, což je opravdu hodně. Rychlejší způsob barvení představíme v dalších sekcí této kapitoly.

5.2 Výpočet barvy pro vrcholy v grafu H

V předchozí sekci časově nejsložitější operací bylo dobré barvení grafu H. Připomeňme, že se jednalo o časovou složitost $170F^2 \in \mathcal{O}(F^2)$. Jedna z možností jak snížit konstantu co možná nejméně, je využít výpočet barvy pro vrchol grafu H. Odpadá zde situace, kdy je nutné procházet sousední vrcholy grafu H a zjišťovat nepoužitou barvu na okolí vrcholu. Nápad může být takový, že první oblast f_i grafu S z dvojice (f_i, f_j), tj. vrchol grafu H, bude určovat začátek rozsahu

barev a druhá oblast f_j z grafu S bude určovat posun v daném rozsahu barev grafu H. Číslo, které určíme na základě oblasti f_i, budeme dále v textu nazývat *color*. Obdobně číslo, které určíme na základě oblasti f_j budeme dále v textu nazývat *shade*. Pro další fází si připomeňme sestavení grafu T ze strany 58. Graf T je grafem sousednosti oblastí v grafu S přes vrcholy. Poznamenejme, že v grafu T nevede žádná hrana do vrcholu sebe sama, tj. graf T neobsahuje smyčky. Nápad na předpis funkce $Z : F_S \to \mathbb{N}$ může být takový, že za funkci Z vezmeme funkci $c : V(T) \to (1, 2, \ldots, \Delta(T) + 1)$ dobrého barvení grafu T. Poznamenejme, že množiny $V(T)$ a F_S budeme brát za identické. Hledanou funkci pro výpočet barvy v grafu H budeme značit λ. Zkusme popsat vlastnosti, které očekáváme od naší funkce λ. Definičním oborem funkce λ by měly být vrcholy grafu H, což jsou uspořádané dvojice oblastí grafu S. Oborem hodnot by měly být čísla z množiny barev dobrého vrcholového barvení grafu H, což víme, že může být maximálně $\Delta(H)+1$ barev z Lemmatu 2 ze strany 22. Navíc budeme muset zavést další funkci Z, která nám každou oblast grafu S převede na celé číslo. Dále symbolem $K \in \mathbb{Z}$ značit konstantu posunu v rozsahu barev grafu H. Naše funkce $\lambda : F_s \times F_s \to \{1, 2, \ldots, \Delta(G) + 1\}$ by mohla mít předpis:

$$\forall f_i, f_j \in F_S : \ \lambda(f_i, f_j) := K \cdot Z(f_i) + Z(f_j). \tag{39}$$

Vraťme se zpátky k naší myšlence s výpočtem barvy v grafu H a vzorci 39. Pokud nyní dosadíme za funkci Z funkci dobrého barvení grafu T a za konstantu K hodnotu $\Delta(T) + 1$ ve vzorci 39, pak obor hodnot funkce λ bude pro nejhorší případ obarvení grafu T rovný $H_\lambda = \{\Delta(T) + 2 = 14, \Delta(T)+3 = 15, \ldots, (\Delta(T)+1)^2+\Delta(T)+1 = 182\}$. Velikost množiny H_λ je $|H_\lambda| = 169$ stejně jako maximální počet barev na dobré obarvení grafu H pomocí Algoritmu 7 ze strany 7. Osobně by se nám líbilo, kdyby obor hodnot byl $\{1, 2, \ldots, (\Delta(T)+1)^2 = 169\}$. To se dá zařídit odečtením jedné konstanty K ve vzorci 39. Navíc, pokud se algoritmu dobrého barvení obarvit graf T pomocí méně barev než je $\Delta(T) + 1$ např. pomocí $k \in \mathbb{N}$ barev, kde $k < \Delta(T) + 1$, pak některé barvy v grafu H nebudou využity pokud funkce c, bude mít obor hodnot $H_c = \{1, 2, \ldots, k\}$. Maximální hodnota v upravené funkci λ při volbě konstanty $K = k$ bude k^2, což je méně než $(\Delta(T) + 1)^2$ Zaveďme nyní již upravenou funkci $\lambda_K : F_S \times F_S \to \{1, 2, \ldots, K^2\}$ s předpisem

$$\forall f_i, f_j \in F_S : \ \lambda(f_i, f_j) := K \cdot (c_T(f_i) - 1) + c_T(f_j), \tag{40}$$

kde funkce c_T je funkce dobrého barvení grafu T a konstantu K budeme uvažovat jako maximální hodnotu barvy, která je přiřazená oblasti $f \in F_S$ ve funkci c_T. Pojďme nyní vyslovit Větu 7 a k ní uveďme důkaz, že tato metoda výpočtu barvy pomocí vzorce 40 dobře obarví graf H. Tento výsledek je dílem autora této práce.

Věta 7 *Předpokládejme, že máme dobře vrcholově obarvený graf T pomocí funkce $c : V(T) \to \mathbb{N}$, navíc graf T vznikl z rovinného grafu S, kde všechny oblasti byly trojúhelníky. Jestliže při barvení grafu H využijeme výpočet barvy pomocí vzorce (40), kde za funkci c ve vzorci 40 vezeme*

funkci c, pak nemůže vzniknout žádná kolize při paralelním zápisu hodnot z lokálních matic do buněk v globální matici $\mathbb{V}$ *v časových intervalech určených barvou ze vzorce (40).*

Důkaz Důkaz rozdělíme na dvě části. V první části se zaměříme na situaci, kdy dvě libovolné různé oblasti f_1 a f_2 grafu S, v našem případě se jedná o vrcholy grafu T, mají přiřazenou stejnou barvu $b_1 \in \mathbb{N}$, tj. $c(f_1) = c(f_2) = b_1$. Další oblast f_3 grafu S, která odpovídá vrcholu grafu T, bude mít barvu $b_2 \in \mathbb{N}$, tj. $c(f_3) = b_2$. Předpokládejme, že $b_1 \neq b_2$ a hodnota konstanty K bude $\Delta(T)+1$. Ukážeme, že nemůže vzniknout kolize při zápisu hodnot z lokálních matic $\mathbb{V}_{loc}^{1,3}$ a $\mathbb{V}_{loc}^{2,3}$ do buněk v matici $\mathbb{V}$. V druhém případě rozebereme případ, kdy oblasti f_1 a f_2 grafu S, tj. vrcholy grafu T, budou mít přiřazenou stejnou barvu b_1 v dobrém barvení grafu T. Zde ukážeme, že nemůže dojít ke kolizi při zápisu hodnot z lokálních matic $\mathbb{V}_{loc}^{1,1}$, $\mathbb{V}_{loc}^{1,2}$, $\mathbb{V}_{loc}^{2,1}$ a $\mathbb{V}_{loc}^{2,2}$. V obou případech využijeme techniku sporem.

Případ 1
Předpokládejme kolizi při zápisu hodnot z lokálních matic $\mathbb{V}_{loc}^{1,3}$ a $\mathbb{V}_{loc}^{2,3}$. Tyto lokální matice můžeme ztotožnit s dvojicemi oblastí (f_1, f_3) a (f_2, f_3), které jsou současně i vrcholy grafu H. Ze vzorce (40) dostáváme pro vrcholy (f_1, f_3) a (f_2, f_3) grafu H následující barvu.

$$\text{vrchol } (f_1, f_3) \text{ má barvu } \lambda_K(f_1, f_3) = (c(f_1) - 1) \cdot K + c(f_3) = (b_1 - 1) \cdot K + b_2,$$
$$\text{vrchol } (f_2, f_3) \text{ má barvu } \lambda_K(f_2, f_3) = (c(f_2) - 1) \cdot K + c(f_3) = (b_1 - 1) \cdot K + b_2$$

Přiřazené barvy vrcholům (f_1, f_3) a (f_2, f_3) grafu H jsou shodné a tedy podle předpokladu mohou být lokální matice $\mathbb{V}_{loc}^{1,3}$ a $\mathbb{V}_{loc}^{2,3}$ sestaveny paralelně ve stejném časovém slotu. My však předpokládáme, že existuje kolize při zápisu hodnot z lokálních matic $\mathbb{V}_{loc}^{1,3}$ a $\mathbb{V}_{loc}^{2,3}$ do buněk v matici $\mathbb{V}$. Vrcholy, které jsou součásti oblasti f_1 v grafu S si označíme v_1, v_2, a v_3. Podobně vrcholy v grafu S, které tvoří oblast f_2 si označíme v_4, v_5, a v_6 a vrcholy, které tvoří oblast f_3 v grafu S, si označme v_7, v_8, a v_9. Sestavme lokální matice $\mathbb{V}_{loc}^{1,3}$ a $\mathbb{V}_{loc}^{2,3}$, viz Tabulky 19 a 16. Připomeňme,

$$\mathbb{V}_{loc}^{1,3} = \begin{pmatrix} \mathbb{V}^{1,3}[1,7] & \mathbb{V}^{1,3}[1,8] & \mathbb{V}^{1,3}[1,9] \\ \mathbb{V}^{1,3}[2,7] & \mathbb{V}^{1,3}[2,8] & \mathbb{V}^{1,3}[2,9] \\ \mathbb{V}^{1,3}[3,7] & \mathbb{V}^{1,3}[3,8] & \mathbb{V}^{1,3}[3,9] \end{pmatrix}$$

Tabulka 15: Lokální matice souřadnic $\mathbb{V}_{loc}^{1,3}$

$$\mathbb{V}_{loc}^{2,3} = \begin{pmatrix} \mathbb{V}^{2,3}[4,7] & \mathbb{V}^{2,3}[4,8] & \mathbb{V}^{2,3}[4,9] \\ \mathbb{V}^{2,3}[5,7] & \mathbb{V}^{2,3}[5,8] & \mathbb{V}^{2,3}[5,9] \\ \mathbb{V}^{2,3}[6,7] & \mathbb{V}^{2,3}[6,8] & \mathbb{V}^{2,3}[6,9] \end{pmatrix}$$

Tabulka 16: Lokální matice souřadnic $\mathbb{V}_{loc}^{2,3}$

že hranatými závorkami ve výrazu $\mathbb{V}^{k,l}[j,i]$ rozumíme jako souřadnice buněk v matici $\mathbb{V}$, kde se zapíše příslušná hodnota. Protože předpokládáme, že existuje kolize při zápisu hodnot z lokálních matic do buněk v matici $\mathbb{V}$, musí proto existovat v lokální matici alespoň jedna souřadnice matice $\mathbb{V}$, která je stejná. Vidíme, že sloupcové souřadnice v globální matici $\mathbb{V}$ jsou pro lokální matice $\mathbb{V}_{loc}^{1,3}$ a $\mathbb{V}_{loc}^{2,3}$ shodné, tj. j-tý index v $[i,j]$, kde $j \in \{7,8,9\}$. Stačí tedy najít společnou řádkovou souřadnici (i-tý index v $[j,i]$) mezi množinami $\{1,2,3\}$ a $\{4,5,6\}$ tak, aby vznikla kolize. To ale není možné, neboť by vrcholy, které tvoří oblasti f_1 a f_2 grafu T, musely mít

v dobrém barvení grafu T různou barvu, jestliže sdílejí některý vrchol. Jedná se tedy o spor s předpokladem, že graf T je dobře vrcholově obarvený

Případ 2
Opět budeme předpokládat kolizi při zápisu hodnot z lokálních matic do buněk v globální matici $\mathbb{V}$. Nyní budeme zkoumat čtyři dvojice lokálních matic $\mathbb{V}_{loc}^{1,1}$, $\mathbb{V}_{loc}^{1,2}$, $\mathbb{V}_{loc}^{2,1}$ a $\mathbb{V}_{loc}^{2,2}$. Tyto lokální matice můžeme ztotožnit s dvojicemi vrcholu $(f_1, f_1), (f_1, f_2), (f_2, f_1)$ a (f_2, f_2) což odpovídá i vrcholům v grafu H. Předpokládáme, že $c(f_1) = c(f_2) = b_1$. Ověřme, zda zápis hodnot z lokálních matic $\mathbb{V}_{loc}^{1,1}, \mathbb{V}_{loc}^{1,2}, \mathbb{V}_{loc}^{2,1}$ a $\mathbb{V}_{loc}^{2,2}$ může být vykonán paralelně. Pokud dvojice oblastí, tj. vrcholy grafu H budou mít vypočtenou stejnou barvu, která vzešla ze vzorce 40. Vypočtěme tedy barvu pro vrcholy $(f_1, f_1), (f_1, f_2), (f_2, f_1)$ a (f_2, f_2) grafu H pomocí vzorce 40.

$$\begin{aligned}
&\text{vrchol } (f_1, f_1) \text{ má barvu } \lambda_K(f_1, f_1) = (c(f_1) - 1) \cdot K + c(f_1) = (b_1 - 1) \cdot K + b_1,\\
&\text{vrchol } (f_1, f_2) \text{ má barvu } \lambda_K(f_1, f_2) = (c(f_1) - 1) \cdot K + c(f_2) = (b_1 - 1) \cdot K + b_1,\\
&\text{vrchol } (f_2, f_1) \text{ má barvu } \lambda_K(f_2, f_1) = (c(f_2) - 1) \cdot K + c(f_1) = (b_1 - 1) \cdot K + b_1,\\
&\text{vrchol } (f_2, f_2) \text{ má barvu } \lambda_L(f_2, f_2) = (c(f_2) - 1) \cdot K + c(f_2) = (b_1 - 1) \cdot K + b_1.
\end{aligned}$$

Vidíme, že všechny čtyři dvojice oblastí, tj. vrcholy grafu H, mají přiřazenou stejnou barvu a mohou tedy zápisy všech hodnot z lokální matice do buněk v globální matici $\mathbb{V}$ vykonány paralelně. My však předpokládáme, že existuje kolize mezi zápisy těchto hodnot do buněk v globální matici $\mathbb{V}$ Připomeňme, že množinu vrcholů grafu S, které tvoří oblast f_1, jsme označili v_1, v_2 a v_3 a množinu vrcholů, které tvoří oblast f_2 jsme označili v_4, v_5 a v_6. Sestavme lokální matice $\mathbb{V}_{loc}^{1,1}, \mathbb{V}_{loc}^{1,2}, \mathbb{V}_{loc}^{2,1}$ a $\mathbb{V}_{loc}^{2,2}$, viz Tabulky 17,18, 19 a 20.

$$\mathbb{V}_{loc}^{1,1} = \begin{pmatrix} \mathbb{V}^{1,1}[1,1] & \mathbb{V}^{1,1}[1,2] & \mathbb{V}^{1,1}[1,3] \\ \mathbb{V}^{1,1}[2,1] & \mathbb{V}^{1,1}[2,2] & \mathbb{V}^{1,1}[2,3] \\ \mathbb{V}^{1,1}[3,1] & \mathbb{V}^{1,1}[3,2] & \mathbb{V}^{1,1}[3,3] \end{pmatrix}$$

Tabulka 17: Lokální matice souřadnic $\mathbb{V}_{loc}^{1,1}$

$$\mathbb{V}_{loc}^{1,2} = \begin{pmatrix} \mathbb{V}^{1,2}[1,4] & \mathbb{V}^{1,2}[1,5] & \mathbb{V}^{1,2}[1,6] \\ \mathbb{V}^{1,2}[2,4] & \mathbb{V}^{1,2}[2,5] & \mathbb{V}^{1,2}[2,6] \\ \mathbb{V}^{1,2}[3,4] & \mathbb{V}^{1,2}[3,5] & \mathbb{V}^{1,2}[3,6] \end{pmatrix}$$

Tabulka 18: Lokální matice souřadnic $\mathbb{V}_{loc}^{1,2}$

$$\mathbb{V}_{loc}^{2,1} = \begin{pmatrix} \mathbb{V}^{2,1}[4,1] & \mathbb{V}^{2,1}[4,2] & \mathbb{V}^{2,1}[4,3] \\ \mathbb{V}^{2,1}[5,1] & \mathbb{V}^{2,1}[5,2] & \mathbb{V}^{2,1}[5,3] \\ \mathbb{V}^{2,1}[6,1] & \mathbb{V}^{2,1}[6,2] & \mathbb{V}^{2,1}[6,3] \end{pmatrix}$$

Tabulka 19: Lokální matice souřadnic $\mathbb{V}_{loc}^{2,1}$

$$\mathbb{V}_{loc}^{2,2} = \begin{pmatrix} \mathbb{V}^{2,2}[4,4] & \mathbb{V}^{2,2}[4,5] & \mathbb{V}^{2,2}[4,6] \\ \mathbb{V}^{2,2}[5,4] & \mathbb{V}^{2,2}[5,5] & \mathbb{V}^{2,2}[5,6] \\ \mathbb{V}^{2,2}[6,4] & \mathbb{V}^{2,2}[6,5] & \mathbb{V}^{2,2}[6,6] \end{pmatrix}$$

Tabulka 20: Lokální matice souřadnic $\mathbb{V}_{loc}^{2,2}$

Předpokládejme, že existuje kolize při zápisu hodnot z matic $\mathbb{V}_{loc}^{1,1}, \mathbb{V}_{loc}^{1,2}, \mathbb{V}_{loc}^{2,1}$ a $\mathbb{V}_{loc}^{2,2}$ do buněk v globální matici $\mathbb{V}$. V prvním případě rozebereme kolize zápisu lokální matice $\mathbb{V}_{loc}^{1,1}$ se zápisem matice $\mathbb{V}_{loc}^{2,2}$. V druhém případě si bez újmy na obecnosti vybereme jednu dvojici lokálních matic, které buď má sloupec nebo řádek matice $\mathbb{V}$ shodný. Aby vznikla kolize během zápisu hodnot z lo-

kálních matic $\mathbb{V}_{loc}^{1,1}$ a $\mathbb{V}_{loc}^{2,2}$, musí být nějaký vrchol z množiny $\{v_1, v_2, v_3\}$ shodný s vrcholem z množiny $\{v_4, v_5, v_6\}$. Pokud by takové situace nastala, musely by vrcholy f_1 a f_2 v grafu T být spojené hranou a tedy mít různou barvu v dobrém vrcholovém barvení, což je spor s předpokladem, že $c(f_1) = c(f_2) = b_1$. Pro druhý případ si vybereme dvojici lokálních matic $\mathbb{V}_{loc}^{1,2}$ a $\mathbb{V}_{loc}^{2,2}$. Taktéž předpokládáme, že vznikne kolize při zápisu hodnot těchto lokálních matic do buněk v globální matice $\mathbb{V}$. Vidíme, že obě lokální matice mají stejné souřadnice sloupců, tj. $\{4, 5, 6\}$. Aby vznikla kolize, musí mít alespoň jednu souřadnici řádků stejnou. Znamená to, že nějaký vrchol z množiny $\{v_1, v_2, v_3\}$ musí být totožný s nějakým vrcholem z množiny $\{v_4, v_5, v_6\}$. Pokud by to tak bylo, dostáváme se do stejné situace jako v předchozím případě, že oblasti f_1 a f_2 v grafu S sdílejí stejný vrchol. Následně v dobrém vrcholovém barvení grafu T by vrcholy f_1 a f_2 musely mít různou barvu, což je spor s předpokladem, že mají stejnou barvu. ■

5.3 Algoritmus dobrého barvení grafu H pomocí výpočtu barvy

V této kapitole popíšeme Algoritmus 9 dobrého barvení grafu H, který určí barvu všem vrcholů H za pomocí výpočtu. Popis Algoritmu 9 rozdělíme na tři sekce. V první sekci se budeme zabývat konstrukci grafu T. V druhé sekci se zaměříme na dobré vrcholové barvení grafu T. V poslední třetí sekci využijeme dobře obarvený graf T k výpočtu barvy ve vrcholech grafu H. Algoritmus 9 je velice podobný s Algoritmem 7, neboť se jedná o vylepšení samotného Algoritmu 7. Proto, některé části nebudou detailně popsány, neboť je jim věnován detailnější popis v kapitole 5 na straně 60. Vylepšením oproti předchozímu Algoritmu 7 je snížení konstanty u vévodící funkce v časové složitosti. Opět i tomto případě vrcholy grafu H budou dvojice oblastí $\tau_j\tau_i$ sítě.

5.3.1 Sestavení grafu T

V této sekci se zaměříme na 7. až 12. řádek v Algoritmu 9. Tato sekce je hodně podobná se sekcí 5.1.3 ze strany 5.1.3 kdy jsme v Algoritmu 7 sestavovali pole *ElementNeigbours*. Zde je jedinný rozdíl ten, že všechny množiny *ElementNeigbors*$[\tau_i]$ neobsahuji v sobě prvek (element) τ_i. Jinak princip sestavení pole *ElementNeighbors* je stejný jako v sekci 5.1.3. Na pole *ElementNeighbors* se můžeme dívat jako na množinovou funkci, která pro zadaný vrchol grafu T vrátí množinu všech sousedních vrcholů. V našem případě jsou vrcholy grafu T elementy sítě.

Časová složitost sestavení pole *ElementNeighbors* je stejná jako v předchozím Algoritmu 7, tj. $3\Delta(S)F = 18F \in \mathcal{O}(F)$. Oproti tomu paměťová náročnost se snížila o konstantu, neboť není potřeba uchovávat pro každý element informaci o elementu samotném v poli *ElementNeighbors*. Paměťová náročnost sestavení pole *ElementNeighbors* je $\Delta(T)F = 12F \in \mathcal{O}(F)$.

5.3.2 Dobré vrcholové barvení grafu T

V této sekci se zaměříme na 13. až 21. řádek v Algoritmu 9. Cílem v této sekci je dobře obarvit graf T, kde vrcholy grafu T jsou oblasti grafu S v našem případě je je grafem S vstupní síť a oblasti grafu S jsou elementy. V předchozí sekci jsme zmínili, že na pole *ElementNeigbors* se můžeme dívat jako na množinovou funkci, která vrátí pro každý vrchol grafu T množinu sousedních vrcholů. Díky tomu, můžeme využít hladový Algoritmus 3 ze strany 26. Opět budeme využívat přítomnost nulové barvy a její význam zůstane zachován. Z Lemmatu 2 ze strany 22, víme že maximální počet barev na dobré obarvení grafu T je dán vzorcem $\Delta(T) + 1 = 12 + 1 = 13$. Z Brooksovy věty 3 ze strany 23 víme, že pokud graf není lichý cyklus nebo kompletní graf pak potom horní odhad počtu barev je $\Delta(T)$. Pokud vyloučíme tyto okrajové případy, určitě bude pro náš graf T platit Brooksova věta 3. Protože však využíváme heuristický algoritmus barvení, který nemusí nalézt $\chi(T)$ barev, předpokládáme, že s přidáním jedné barvy k $\Delta(T)$ se mu graf T podaří dobře obarvit. Navíc budeme požívat nulovou barvu, což obnáší zvýšit rozsah o jedničku v poli *usedColor* na hodnotu 14. Samotný algoritmus barvení sestaví pro každý element množinu sousedních elementů pomocí pole *ElementNeighbors*. Tuto množinu sousedních vrcholů algoritmus využije při sestavení pole *usedColor*, ve kterém budeme hledat nejnižší hodnotu nenulové barvy, kterou je možné obarvit vrchol grafu T, tj. element sítě. Na 22. řádku Algoritmu 9 máme v poli *ColorElements* uložené hodnoty barev pro každý element sítě. Na pole *ColorElements* se můžeme dívat jako na barvící funkci c, která každému vrcholu z grafu T přiřadí právě jednu hodnotu barvy z dobrého barvení grafu T, tj. $c : F_s \to \mathbb{N}$. Množinu oblastí F_s s množinou vrcholů grafu T, tj. V_T, považujeme za ekvivalentní. Připomeňme, že v našem případě jsou oblasti grafu S elementy sítě.

Časová složitost sestavení pole *ColorElements* je $\Delta(T)F = 39F \in \mathcal{O}(F)$. Protože, pro každý element je potřeba projít až 12 sousedních elementů, které se nachází na okolí právě barveného elementu, dále metoda *setAllCellsToFalse* nastaví pro každý vrchol grafu T všem 14 vrcholům hodnotu *False* v poli *usedColor*. V metodě *findFirstFalse* se v nejhorším případě projde 13 buněk v poli *usedColor* (nepřistupujeme k nulové barvě). Množiny sousedních elementů ke každému vrcholu jsou uložené v poli *ElementNeighbours*. Paměťová složitost sestavení pole *ElementNeighbours* je $\mathcal{O}(F)$, neboť je potřeba si uložit barvu každému elementu v poli *ColorElements*.

5.3.3 Dobré vrcholové barvení grafu H metodou výpočtu barvy

V této kapitole se zaměříme na zbývající řádky v Algoritmu 9, tj. 22. až 28. řádek. Cílem v této sekci je popsat barvení grafu H, pomocí výpočtu, který jsme uvedli ve vzorci 40 na straně 65. Jak jsme již v předchozí podkapitole nastínili, za funkci g vezmeme funkci $c - 1$, kde funkce c je funkcí dobrého barvení grafu T. V našem případě diskrétní funkci c, reprezentujeme pomocí

pole *ColorElements*. Díky vlastnostem sítě a odpovídajícího grafu T, nastavíme konstantu K na hodnotu 13. Připomeňme, že dobře barvíme vrcholy graf H, které reprezentujeme pomocí dvojic elementů. Proto, je potřeba projít všechny uspořádané dvojice a na základě vzorce 40 přiřadit každému vrcholu grafu H barvu do matice *ColorMatrix*.

Časová složitost sestavení matice *ColorMatrix* je $340F^2 \in \mathcal{O}(F^2)$, protože je potřeba projít všechny uspořádané dvojice oblastí, kterých je F^2. Složitost samotného výpočtu bereme za konstantní. Složistost metody *setAllCellsToFalse* je nastavit pro každý vrchol v grafu H 170 buněk na hodnotu *False* a při metodě *findFirstFalseIndex* se v nejhorším případě přistoupí k 169 buňkám. Paměťová složitost sestavení matice *ColorMatrix* je $F^2 \in \mathcal{O}(F^2)$, protože je potřeba uložit barvu každé uspořádané dvojici elementů.

Poznámka 15 Po ukončení Algoritmu 9 bude potřeba využít Algoritmus 8, který se nachází na straně 64. Tento algoritmus z matice *ColorMatrix* vytvoří barevné třídy, které využijeme při paralelním sestavení matice $\mathbb{V}$ v metodě hraničních prvků.

5.3.4 Časová a paměťová složitost Algoritmu 9

Celková časová složitost algoritmu je dána součtem časových složitostí v jednotlivých sekcích. Nesmíme však zapomenout na časovou složitost sestavení pole *NodeElements*, které jsme v této kapitole nezmiňovali, protože je již popsané v kapitole 5.1.2 na straně 61. Dostáváme tak celkovou časovou složitost, která je $3F + 18F + 12F + F^2 = 33F + 340F^2 \in \mathcal{O}(F^2)$.

Celková paměťová složitost algoritmu je dána součtem paměťových složitostí v jednotlivých sekcích. Opět nesmíme zapomenout na paměťovou složitost sestavení pole *NodeElements*. V součtu pak obdržíme celkovou paměťovou náročnost, která je $6N+12F+F+F^2 = 16F-4+F^2 \in \mathcal{O}(F^2)$. V odhadu $6N$ jsme využili Důsledek 1 ze strany 19.

5.3.5 Shrnutí

V Algoritmu 9 se nám podařilo snížit konstantu o 168 u vévodícího členu z 508 na 34O. Což je dobrý krok. Při použití bitové reprezentace čísel a vhodného algoritmu, lze časová složitost metod *setAllCellsToFalse* a *findFirstSet* snížit na minimum. Bohužel však máme problém s tím, že časová i paměťová náročnost je $\mathcal{O}(F^2)$. V následující kapitole si představíme trik, pomocí kterého sestavíme určení barevných tříd grafu H s časovou ale i paměťovou složitostí $\mathcal{O}(F)$ navíc využijeme faktu, že při sestavování matice $\mathbb{V}$ se iteruje přes uspořádané dvojice elementů.

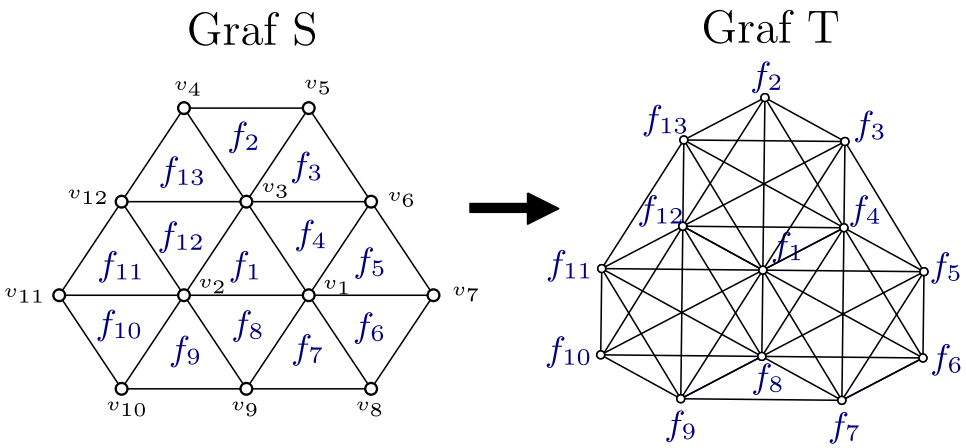
Algorithm 9 Výpočet barvy v grafu H

Require: nodes count N, elements count F, Set of elements $\{\tau\}$

NodeElemtents $\leftarrow$ ArraySet[N][6]
for all element $\tau_i \in \{\tau\}$ **do**
    for all node $x_j \in \tau_i$ **do**
        NodeElemtents[x_i] $\leftarrow$ NodeElemtents[x_j] $\cup$ $\{\tau_j\}$
    end for
end for
ElementNeighbors $\leftarrow$ ArraySet[F][12]
for all element $\tau_i \in \{\tau\}$ **do**
    for all node $x_j \in \tau_i$ **do**
        ElementNeighbors[τ_i] $\leftarrow$ (ElementNeighbors[τ_i] $\cup$ NodeElemtents[x_j]) $\setminus$ $\{\tau_i\}$
    end for
end for
ColorElements $\leftarrow$ ZeroArray[F]
usedColors $\leftarrow$ BoolArray[14]
for all element $\tau_i \in \{\tau\}$ **do**
    usedColors.setAllCellsToFalse()
    for all element $\tau_j \in$ ElementNeighbors[τ_i] **do**
        usedColor[ColorElements[τ_j]]$\leftarrow$ True
    end for
    ColorElements[τ_i] $\leftarrow$ usedColors.findFirstFalseIndex()
end for
ColorMatrix $\leftarrow$ Zero2DArray[F][F]
K = 13
for all element $\tau_i \in \{\tau\}$ **do**
    for all element $\tau_j \in \{\tau\}$ **do**
        ColorMatrix[τ_i][τ_j] $\leftarrow$ K $\cdot$ (ColorElements[τ_i]$-$1) + ColorElements[τ_j]
    end for
end for

5.4 Snížení počtu barev v Algoritmu 9

V této sekci provedeme analýzu dobrého vrcholového barvení grafu H v Algoritmu 9. Dále provedeme rozbor grafu T a stanovíme minimální počet barev na dobré obarvení grafu H. Heuristický Algoritmus 9 nemusí vždy najít $\chi(T)$ obarvení grafu T, ale také nemusí využít všech 13 barev. Pokud Algoritmus 9 nalezne méně barev, budeme moct upravit konstantu K a ušetřit výpočetní čas přes barvy, které jsou prázdné. Připomeňme si znovu Obrázek 20 sestavení grafu T.
Na Obrázku 20 můžeme vidět, že v grafu T vznikají kliky o velikosti alespoň 6, tj. $\omega(T) \geq 6$. Jedná se o vrcholy z množin $\{f_1, f_2, f_3, f_4, f_{12}, f_{13}\}$, $\{f_1, f_4, f_5, f_6, f_7, f_8\}$ a $\{f_1, f_8, f_9, f_{10}, f_{11}, f_{12}\}$ grafu T. Pomocí Lemmatu 1 ze strany 22 víme, že dolní odhad chromatického čísla grafu T je alespoň 6, tj. $\chi(T) \geq \omega(T) \geq 6$. Tato vlastnost se dá dále zobecnit ve Větě 8, ke které uvedeme i důkaz. Jedná se o vlastní důkaz autora této práce.



Obrázek 20: Sestavení grafu T, který odpovídá triangulaci

Věta 8 *Máme-li v rovinném graf S vrchol stupně k takový, že je je součástí k oblastí, potom klikové číslo grafu T, který vznikl operací $\eta(S)$, je alespoň k, tj. $\omega(T) \geq k$.*

Důkaz Zvolme si libovolný, ale pevný vrchol stupně k, který je součástí k oblastí v grafu S, označme jej v. Oblasti, ve kterých je daný vrchol v součásti, si označíme symboly f_1, f_2, …, f_k. Zvolme libovolnou oblast z množiny oblastí $\{f_1, f_2, \ldots, f_k\}$ např. oblast f_i, kde $i \in \{1, 2 \ldots, f_k\}$. Protože oblast f_i sdílí s oblastmi z množiny $\{f_1, f_2, \ldots, f_k\}\backslash\{f_i\}$ vrchol v v grafu S, bude vrchol f_i v grafu T spojený hranami se všemi vrcholy z množiny $\{f_1, f_2, \ldots, f_k\}\ \backslash\{f_i\}$. Tímto způsobem přidáme $k-1$ hran začínajících ve vrcholu f_i a končících ve vrcholech z množiny $\{f_1, f_2, \ldots, f_k\}\backslash\{f_i\}$ v grafu T. Pokud tento způsob uděláme pro všechny vrcholy f_i, kde $i \in \{1, 2, \ldots\ k\}$. Přidáme tak do grafu T za každou z k oblastí $k-1$ hran mezi vrcholy $\{f_1, f_2, \ldots, f_k\}$. Protože jsme každou hranu započetli dvakrát, je potřebné výsledný počet hran podělit dvěma. Dostáváme tak $\frac{k(k-1)}{2}$ hran. Jediný podgraf grafu T, který má $\frac{k(k-1)}{2}$ hran mezi k vrcholy, je klika o velikosti k na okolí vrcholu v a tedy $\omega(T) \geq k$. ∎

Díky Větě 8 víme, že minimální počet barev na dobré vrcholové barvení grafu H je alespoň 36. Čtenář může nabýt dojmů, že samotné barvení zvětšuje počet množin, ve kterých může být problém s kolizemi vyřešen. Pojďme se podívat na celkový rozbor úlohy s kolizemi. Řád matice $\mathbb{V}$ je N^2, kde označením N rozumíme počet vrcholů grafu S, tj. počet uzlů sítě. Dále víme, že v metodě hraničních prvků je třeba vypočítat F^2 lokálních matic pro sestavení globální matice $\mathbb{V}$. Označení F rozumíme počet oblastí grafu S, tj. počet elementů sítě. Za každou sestavenou lokální matici je potřeba přistoupit k devíti buňkám v matici $\mathbb{V}$ při použití po částech lineárních bázových funkcí. Celkem tedy algoritmus během sestavení matice $\mathbb{V}$ v metodě hraničních prvků přistoupí k $9 \cdot F^2$ buňkám v matici $\mathbb{V}$. Maximální počet přístupu v jedné nezávislé množině, ve které nemůže vzniknout kolize, je dán řádem matice $\mathbb{V}$, tj. N^2. Minimální počet časových

množin $\mathcal{X}$, ve kterých by nevznikla kolize je dán vztahem.

$$\mathcal{X} = \left\lceil \frac{9 \cdot F^2}{N^2} \right\rceil$$

Protože předpokládáme, že náš graf S je rovinný a síť 3D objektu úlohy je pokrytá pouze trojúhelníky, můžeme proto počet časových množin $\mathcal{X}$ upřesnit podle Důsledku 1 ze strany 19. Z Důsledku 1 vyplývá, že počet oblastí v rovinném grafu s trojúhelníky je $F = 2N - 4$.

$$\mathcal{X} = \left\lceil \frac{9 \cdot F^2}{N^2} \right\rceil = \left\lceil \frac{9 \cdot (2N-4)^2}{N^2} \right\rceil = \left\lceil \frac{9 \cdot (4N^2 - 16N + 16)}{N^2} \right\rceil = \left\lceil 36 - \frac{144}{N} + \frac{144}{N^2} \right\rceil$$

Pokud bude mít graf S více než 143 vrcholů, dostáváme $\mathcal{X} = 36$ nejmenší možný počet množin. Můžeme vidět, že i analytickým výpočtem jsme dospěli k počtu 36 nezávislých množin pro náš příklad.

5.4.1 Úprava rozsahu počtu barev v Algoritmu 9

Od této chvíle víme, že pokud budeme barvit trojúhelníkovou síť, budeme vždy potřebovat alespoň šest barev na dobré vrcholové barvení grafu T. Pokud část Algoritmu 9, která je zodpovědná za barvení grafu T, najde k dobrému vrcholovému barvení méně barev než třináct, což je horní odhad počtu barev, můžeme upravit konstantu K v Algoritmu 9 na hodnotu maximálního počtu nalezených barev v dobrém vrcholovém barvení grafu T. Pokud použijeme 10 barev k dobrému vrcholovému barvení grafu T místo horního odhadu 13, budeme v další části Algoritmu 9 potřebovat $10 \cdot 10 = 100$ barev k dobrému vrcholovému barvení grafu H místo $13 \cdot 13 = 169$ barev, což je velká úspora. V Algoritmu 9 to znamená vytvořit celočíselnou proměnnou *maxColorValue* a při každém přiřazení nové barvy grafu T testovat, zda se nezvýšila aktuální hodnota barvy a změnit hodnotu proměnné *maxColorValue* na tuto hodnotu. Po ukončení dobrého vrcholového barvení grafu T nastavíme konstantu K na hodnotu proměnné *maxColorValue*.

5.5 Vylepšení algoritmů dobrého vrcholového barvení grafu H

V předchozích sekcích jsme ukázali vlastní Algoritmus 7 na straně 63 a jeho vylepšení v podobě Algoritmu 9 na straně 71. Velkou nevýhodou obou zmíněných algoritmů je velká časová a paměťová složitost, která v obou dvou případech je $\mathcal{O}(F^2)$. Posledním nápadem, jak snížit tuto časovou a paměťovou složitost radikálně, je myšlenka sestavit předpis pro vznik barevných tříd grafu H, bez nutnosti barvení grafu H. Potom by ve smyčce metody hraničních prvků probíhalo něco, co by se dalo nazvat sestavení barevných tříd grafu H. Samotná myšlenky čerpá z Algoritmu 9 a poznatku, že je možné stanovit všechny barevné třídy grafu H za předpokladu, že je

dobře vrcholově obarven graf T bez toho, aniž bychom barvili graf H. Tuto myšlenku detailně rozebereme. Základním předpokladem je mít dobře vrcholově obarvený menší graf T. V Algoritmu 9 na straně 71 uchováváme barvy jednotlivých vrcholů grafu T v poli *ColorElements*. Pokud bychom vytvořili nové pole množin *ElementsInColor*, které by obsahovalo barevné třídy grafu T, pak bychom mohli zjistit barvu každé dvojice interagující oblastí v metodě hraničních prvků. Popis vzniku pole *ElementsInColor* je popsán Algoritmem 10. Vstupem Algoritmu 10 je jednorozměrné pole *ColorElements*, kde již jsou obarvené vrcholy grafu T. Cílem je seskupit jednotlivé obarvené elementy do množin podle stejné barvy. Výstupem Algoritmu 10 je pole *ElementsInColor*, tj. barevné třídy grafu T.

Algorithm 10 Zpracování pole *ColorElements*

Require: array ColorElements, elements count F, set of elements $\{\tau\}$ maxColorValue

```
1: ElementsInColor ← ArraySet[maxColorValue]
2: for τ_j ∈ {τ} do
3:     ElementsInColor[ColorElements[τ_j]] ← ElementsInColor[ColorElements[τ_j]] ∪{τ_j}
4: end for
```

Nyní máme jednotlivé barevné třídy grafu T uložené v poli *ElementsInColor*. Pomocí vzorce (40) ze strany 65 umíme určit barevné třídy grafu H. Pro lepší pochopení určení konkrétní barevné třídy grafu H za pomocí pole *ElementsInColor* uvedeme příklad.

Příklad 14

Naším cílem je určit všechny vrcholy konkretní barvy v dobrém vrcholovém barvení grafu H bez toho, aniž bychom barvili graf H. Jeden z předpokladů je mít dobře obarven graf T. Předpokládejme, že máme dobře vrcholově obarven menší graf T pomocí 10 barev. Zajímají nás barevná třída 58 grafu H, tj. kdy všechny vrcholy grafu H mají přiřazenou barvu 58 v dobrém barvení grafu H. Celkový počet barev grafu H je $10 \cdot 10 = 100$. Uvažujme následující množinu barev $B_T = \{1, 2, \ldots, 10\}$ dobrého vrcholového barvení grafu T. Dále uveďme funkci $c : V(T) \to B_T$, která je realizována v Algoritmu 9 pomocí pole *ColorElements*. Dále předpokládejme, že již máme zpracované barevné třídy grafu T v poli *ElementsInColor*. Například prvky pole uložené v řádku *ElementsInColor[1]* odpovídají indexům oblastí grafu S, tj. indexy vrcholů grafu T, které mají přiřazenou barvu 1 v dobrém vrcholovém barvení grafu T. Konstanta K bude nastavena na hodnotu 10, neboť se nám podařilo graf T dobře vrcholově obarvit 10 barvami. K získání barevné třídy 58 využijeme vzorec (40) ze strany 65. Ve vzorci (40) za funkci c_T použijeme funkci dobrého barvení grafu T. Nyní vyjádříme upravenou rovnici (40) pro barevnou třídu 58.

$$\begin{aligned} 58 &= c_T(j) \cdot 10 + c_T(i) \\ 58 &= c_T(j) - 1) \cdot 10 + c_T(i) \\ 68 &= c_T(j) \cdot 10 + c(i) \end{aligned}$$

Takto upravená rovnice má pro zadané parametry $c(j)$ a $c(i)$ jediné řešení, kde vrcholy i mají barvu 6 a vrcholy j mají barvu 8. Všechny vrcholy které v dobrém vrcholovém barvení barvu 58 grafu H, vzniknou pomocí kartézského součinu všech prvků uložených v řádcích pole *ElementsInColor[6]* a *ElementsInColor[8]*. Připomeňme, že výpočty na uspořádaných dvojicích elementů sítě, které reprezentujeme pomocí vrcholů grafu H, se stejnou barvou v dobrém vrcholovém barvení grafu H, mohou být vykonávaný paralelně v metodě hraničních prvků. Což je i náš případ. Uspořádané dvojice oblastí, tj. vrcholy grafu H, které vznikly kartézským součinem *ElementsInColor[6]* a *ElementsInColor[8]*, mají stejnou barvu 58 v dobrém vrcholovém barvení grafu H a tedy výpočty pro tyto dvojice mohou být vykonávány paralelně v metodě hraničních prvků.

V Příkladu 14 jsme ukázali, jak sestrojit jednu konkrétní barevnou třídu grafu H a to za pomocí barevných tříd grafu T, tj. pole množin *ElementsInColor*. Připomeňme, že jsme ve výsledku určení vrcholů ze stejné barevné třídy použili kartézský součin mezi dvěma barevnými třídami grafu T. To je velmi důležité pozorování, neboť díky tomu můžeme stanovit všechny vrcholy v barevných třídách dobře obarveného grafu H a to tak, že uděláme kartézské součiny mezi všemi barevnými třídami grafu T. Protože každý vrchol grafu T, tj. element sítě, musí mít přiřazenou barvu v poli *ColorElements* a následně zařazen v nějaké barevné třídě grafu T tj. v poli *ElementsInColor*. Když tedy pak vytvoříme kartézské součiny mezi barevnými třídami grafu T, obdržíme všechny dvojice elementů, tj. všechny vrcholy grafu H, které mají přiřazenou barvu na základě dvojice barev grafu T z kartézského součinu. Ukažme samotné použití barevných tříd grafu T při sestavení matice $\mathbb{V}$. Připomeňme, že pro lokální matice, které jsou sestaveny z dvojic elementů stejné barvy, máme zaručeno, že zápis hodnot z těchto lokálních matic do buněk globální matice $\mathbb{V}$ je bezkolizní a navíc může být prováděn paralelně. Pro úplnost zde uvedeme funkční Algoritmus 12, který sestaví barevné třídy grafu T. Algoritmus 12 vychází z Algoritmu 7 ze strany 63, Algoritmu 9 ze strany 71 a Algoritmu 10 ze strany 74. V Algoritmu 12 se nenachází sekce pro uložení a zpracování dobrého vrcholového barvení grafu H, která měla v předchozích algoritmech největší rostoucí funkci v časové, ale i paměťové složitosti. Detailní rozebraní jednotlivých sekcí je provedeno u Algoritmu 7 a Algoritmu 9, ze kterých tento Algoritmus 12 vychází. Časová a paměťová složitost Algoritmu 12 je $\mathcal{O}(\Delta(T)|F|)$, kde $|F|$ je počet oblastí grafu S, nebo též elementů sítě.

5.6 Shrnutí

V této kapitole jsme představili funkční Algoritmus 12, který řeší problém s kolizemi. Myšlenka barvení grafu H není přímo použita, za to se však využilo dobrého vrcholového barvení grafu T. Z dobrého barvení grafu T můžeme snadno zkonstruovat barevné třídy grafu H. Připomeňme si, že výpočty na dvojicích elementů, tj. vrcholy grafu H ze stejné barevné třídy mohou být vykonávány paralelně během sestavení matice $\mathbb{V}$ při výpočtu Laplaceovy rovnice metodou hraničních prvků.

Algorithm 11 [11] Paralelní sestavení matice $\mathbb{V}_h$ v metodě hraničních prvků

Require: ElementsInColor, colorMaxValue

```
1: V_h = Zero2DArray[N][N]
2: V_{h,loc} = Zero2DArray[3][3]
3: kNodesIndex = Zero1DArray[3]
4: lNodesIndex = Zero1DArray[3]
5: for all color ∈ {1, 2, ..., colorMaxValue} do
6:     for all shade ∈ {1, 2, ..., colorMaxValue} do
7:         #pragma paralell for schedule (auto)
8:         for all element τ_k ∈ ElementsInColor[color] do
9:             kNodesIndex ← getNodesIndexFromElement(τ_k)
10:            for all τ_l ∈ ElementsInColor[shade] do
11:                lNodesIndex ← getNodesIndexFromElement(τ_l)
12:                V_{h,loc} ← computeVhLoc(kNodesIndex, lNodesIndex)
13:                for all index i ∈ {1, 2, 3} do
14:                    for all index j ∈ {1, 2, 3} do
15:                        V_h[knodeIndex[i]][lnodeIndex[j]] += V_{h,loc}[i][j]
16:                    end for
17:                end for
18:            end for
19:        end for
20:    end for
21: end for
```

Tím, že jsou vykonávány pro dvojice elementů, tj. vrcholy grafu ze stejné barevné třídy, je zaručeno, že nemůže dojít ke kolizím při zápisu do paměti. Náš navržený postup nevyužívá podporu tzv. atomické operace. Časová náročnost Algoritmu 6 určení barevných tříd grafu H je $\mathcal{O}(\Delta(T)F)$, kde F je počet elementů sítě 3D objektu. V našem případě, kdy nám pro náš případ sítě vychází $\Delta(T) = 12$, můžeme brát, že časová složitost Algoritmu 6 je lineární vzhledem k počtu elementů sítě, tj. $\mathcal{O}(F)$. V další kapitole 6 si představíme další vylepšení Algoritmu 12.

Algorithm 12 Předpřipravené pole *ElementsInColor* k určení barevnách tříd grafu H

Require: nodes count N, elements count F, Set of elements $\{\tau\}$

NodeElemtents ← ArraySet[N][6]
for all element $\tau_i \in \{\tau\}$ **do**
    for all node $x_j \in \tau_i$ **do**
        NodeElemtents[x_i] ← NodeElemtents[x_j] ∪ $\{\tau_j\}$
    end for
end for
ElementNeighbors ← ArraySet[F][13]
for all element $\tau_i \in \{\tau\}$ **do**
    for all node $x_j \in \tau_i$ **do**
        ElementNeighbors[τ_i] ← (ElementNeighbors[τ_i] ∪ NodeElemtents[x_j]) \ $\{\tau_i\}$
    end for
end for
maxColorValue ← 6
ColorElements ← ZeroArray[F]
usedColors ← BoolArray[14]
for all element $\tau_i \in \{\tau\}$ **do**
    usedColors.setAllCellsToFalse()
    for all $\tau_j \in$ ElementNeighbors[τ_i] **do**
        usedColors[ColorElements[τ_j]] ← *True*
    end for
    ColorElements[τ_i] ← usedColors.findFirstFalseIndex()
    maxColorValue ← max(maxColorValue, ColorElements[τ_i])
end for
ElementsInColor ← ArraySet[maxColorValue]
for element $\tau_i \in \{\tau\}$ **do**
    ElementsInColor[ColorElements[τ_j]] ← ElementsInColor[ColorElements[τ_j]] ∪ $\{\tau_j\}$
end for

6 Vylepšení současného řešení

V sekci 5.5 jsme ukázali funkční Algoritmus 12 na straně 77. Jistou nevýhodou může být neoptimální počet použitých barev k dobrému vrcholovému obarvení grafu T a následně s tím související vyšší počet barev v dobrém vrcholovém barvení grafu H. Cílem, který bychom si přáli dosáhnout, je použít co možná nejmenšího počtu možných barev v dobrém vrcholovém barvení grafu T, který způsobí, že budeme mít více vrcholů stejné barvy. S tím souvisí i efektivnější využití paralelismu nad větším počtem nezávislých výpočtů v metodě hraničních prvků. Další věcí je samotné vyvážení počtu vrcholů v barevných třídách dobrého vrcholového barvení grafu T. Důvodem, proč se zabývat samotným vyvážením počtu vrcholů v barevných třídách, je rovnoměrné rozdělení velikostí výpočtů v metodě hraničních prvků během paralelního vykonávání.

6.1 Využití bitových operací

Jednou z možností, jak zrychlit Algoritmus 12 na straně 77, je využít bitové reprezentace čísel. Jedno z míst Algoritmu 12, kde je možné použít bitových operací, je náhrada pole *usedColors* za stejnojmennou strukturu, která v sobě bude uchovávat 32 bitové číslo a implementovat patřičné metody k manipulaci s daným číslem. Pomocí pozice bitů budeme reprezentovat přítomnost jednotlivých barev na okolí konkrétního vrcholu v grafu T. Předpokládáme, že 32 bitů bude dostatečná velikost pro uložení až 32 barev na okolí libovolného vrcholu v grafu T. Horní hranice počtu barev v dobrém vrcholovém barvení grafu T je 13. Podle Brooksovy Věty 3 by mělo stačit $\Delta(T) = 12$ barev a algoritmu $\Delta(T)+1 = 13$ barev. Nyní zobrazme Výpis 1 v jazyce C++ struktury *BitMasking*. Proměnná *usedColors* bude typu *BitMasking* v Algoritmu 12.

```
struct BitMasking{
    int32_t word;
    BitMasking() {
        this->word = 1;
    }
    void clear() {
        this->word = 1;
    }
    void setBit(const int32_t n) {
        this->word |= 1 << n;
    }
    int findFirstFalse() {
        this->word = ~this->word;
```

```
        return ffs(this->word)-1;
    }
};
```

Výpis 1: Struktura BitMasking v jazyce C++

Jak je zmíněno ve Výpisu 1, proměnná *word* bude datového typu int, tj. celé číslo. V konstruktoru *BitMasking()* je proměnná *word* nastavená na výchozí hodnotu 1, neboť víme, že barva s hodnotou 0 a tedy i bit na 0. pozici nebude nikdy použitý, proto 0. bit nastavíme na 1, což odpovídá celočíselné hodnotě 1. Obdobnou funkci bude mít i metoda *clear()*, která přepíše proměnnou *word* na výchozí hodnotu 1. Tato metoda bude obdobou metody *setAllCellsToFalse*. Metoda *setAllCellsToFalse* v Algoritmu 12 měla za úkol nastavit všem prvkům v poli *usedColors* hodnotu *False*, tj. na výchozí hodnotu pole a následně se 0. index pole *usedColors* nastavil na hodnotu *true*. Tyto dvě operace v Algoritmu 12 jsou spojené do jedné metody *clear()* ve struktuře *BitMasking*. Zde můžeme vidět časové zrychlení, že již není potřebné procházet celé pole, ale jedním přiřazením nastavíme proměnou *word* na výchozí hodnotu. Další metodou ve struktuře *BitMasking* je metoda *setBit(const int32_t n)*, jejíž úkolem je nastavit n. bit v proměnné *word* na hodnotu 1 ve smyslu n. použité barvy na okolí konkrétního vrcholu. Poslední metodou, kterou ve struktuře *BitMasking* máme, je metoda *findFirstFalseBit*. Úkolem metody *findFirstFalseBit* je nalézt první nepoužitý bit, který má v proměnné *word* nastavenou hodnotu 0. K tomu algoritmus využije operací dvojkového doplňku, který jedničky vymění za nuly a nuly za jedničky v bitové reprezentaci proměnné *word* a následně využije vestavěné funkce v jazyce C++ *ffs* (*find first set*). Protože před operací dvojkového doplňku jsme v čísle měli hledat první bit, který má nastavenou hodnotu 0, tak po provedení dvojkového doplňku budeme hledat první bit, který má nastavenou hodnotu 1. Proto algoritmus využije výše zmíněnou funkci *ffs*, která vrací index prvního bitu nastaveného na hodnotu 1. Protože funkce má jiné číslování bitů, je proto nutné upravit korekci na správné hodnoty odečtením čísla 1.

6.2 Heuristika barvení

Barvící Algoritmus 12 ze strany 77 slouží k dobrému obarvení grafu T. Funguje na principu přiřazení nejmenší možné barvy, která není na okolí konkretního vrcholu použita. Dalším postupem, který může snížit počet barev v dobrém vrcholovém barvení grafu T, je využít samotnou strukturu grafu S, neboť priorita obarvování byla dosud ponechána na samotném meshovacím algoritmu v metodě hraničních prvků. Samotná myšlenka použití nejmenší nepoužité barvy na okolí konkretního vrcholu grafu T zůstane zachována, změní se pořadí barvení vrcholů. K tomu využijeme upravený algoritmus prohledávání do šířky (BFS). Předpokladem upraveného algoritmu prohledávání do šířky bude souvislost vstupního grafu T. Pokud by vstupní graf T nebyl souvislý, je potřeba dobře vrcholově obarvit každou z komponent souvislosti. My však před-

pokládejme, že vstupní síť 3D objektu je souvislá a tedy bude souvislý i vytvořený graf T. Samotné pořadí vrcholů vstupujících k obarvení bude určeno na základě vzdáleností od prvního obarveného vrcholu v_1 v grafu T, který bude mít index rovný 1 v množině elementů $\{\tau\}$.

6.2.1 Popis Algoritmu 13 využívající lokální strukturu grafu S

Podívejme se na Algoritmus 13, ve kterém využijeme strukturu *BitMasking()* ze sekce 6.1 ze strany 78. Cílem algoritmu je dobře vrcholově obarvit graf T. Na konci Algoritmu 13 budeme mít dobře obarvený graf T, kde informace o přiřazených barvách budou uloženy v poli *ColorElements*. Jak již bylo zmíněno, Algoritmus 13 bude barvit vrcholy v pořadí určené vzdáleností od vrcholu v_1 grafu T pomocí Algoritmu prohledávaní do šířky (BFS). V našem případě se prvním obarvovaným vrcholem stává element sítě τ_1. Tento způsob v sobě zahrnuje vytvořit pole stavů *State*, ve kterém si Algoritmus bude uchovávat informaci o vrcholech. Budeme rozlišovat celkem tři stavy.

- Stav 0 - Element ve stavu 0 zatím nebyl navštíven a tudíž ani zařazen do fronty *processQueue*. Nemá proto zatím přiřazenou žádnou barvu z dobrého barvení grafu T, tzn. má přiřazenou nulovou barvu.
- Stav 1 - Element ve stavu 1 byl navštíven a nachází ve frontě *processQueue*, kde čeká na okamžik kdy na něj přijde řada a bude obarven. Element ve stavu 1 má stále nulovou barvu v poli *ColorElements*.
- Stav 2 - Element ve stavu 2 byl obarven v minulosti a má tedy přiřazenou barvu v poli *ColorElements*.

Princip algoritmu je dále intuitivní, předpokládáme že čtenář je obeznámen s tímto základním algoritmem prohledávání grafu do šířky, tzv. BFS Algoritmem. Zmíníme, že se Algoritmu BFS někdy přezdívá *Flood fill* algoritmus díky své vizuální interpretaci, to znamená rozlévání tekutiny v prostoru. Hlavním důvodem vzniku samotného Algoritmu 13 je heuristicky snížit počet barev v dobrém vrcholovém barvení grafu T.

Časová složitost Algoritmu BFS je $\mathcal{O}(m+n)$, kde m je počet hran grafu T a n počet vrcholů grafu T. V našem případě předpokládejme, že maximální stupeň vrcholů v grafu T je 12 (pouze pro síť kde každý uzel je součástí nejvýše 6 elementů). Potom využijeme princip sudosti, který říká, že pokud sečteme všechny stupně grafu a vydělíme tento součet dvěma, obdržíme počet hran v grafu. V našem případě obdržíme, že náš graf T má maximálně $6F$ hran, kde F je počet oblastí grafu S, nebo též počet elementů sítě. Počet vrcholů v grafu T je F. Celková časová složitost tak je $\mathcal{O}(6F+F) = \mathcal{O}(7F)$, tj. $\mathcal{O}(F)$. Paměťová složitost Algoritmu 13 je $2F \in \mathcal{O}(F)$, neboť vytváříme pole *ColorElements* a pole *State*. Paměťová velikost každého pole je F.

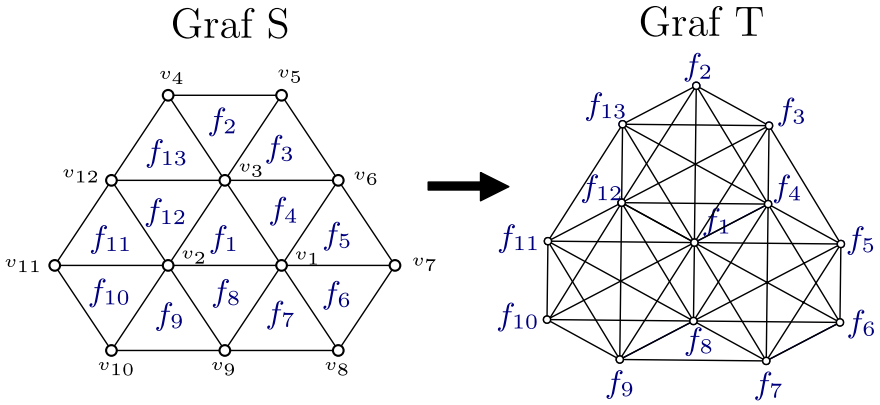
Algorithm 13 Využití struktury grafu v dobrém vrcholovém barvení grafu T

Require: ArraySet ElementNeighbors, start element τ_1, count of elements F

```
1: maxColorValue ← 6 #  inicializace  ω(T) ≥ 6
2: ColorElements ← ZeroArray[F]
3: usedColors ← BitMasking()
4: State ← ZeroArray[F] # default state 0 - unvisited vertex
5: processQueue ← Queue()
6: processQueue.push(τ1)
7: State[τ1] ← 1 # state 1 - waiting for visit
8: while !processQueue.empty() do
9:     τi ← processQueue.pop()
10:    usedColors.clear()
11:    for all τj ∈ ElementNeighbors[τi] do
12:        usedColors.setBit(ColorElements[τj])
13:        if State[τj] == 0 then
14:            State[τj] ← 1
15:            processQueue.push(τj)
16:        end if
17:    end for
18:    ColorElements[τi] ← usedColors.findFirstFalse()
19:    maxColorValue = max(ColorElements[τi], maxColorValue)
20:    State[τi] ← 2 # state 2 - visited vertex
21: end while
```

6.2.2 Popis vylepšeného Algoritmu 13 využívající lokální struktury grafu S

Dalším typem vylepšení je vytvoření pro každou oblast grafu S informaci o sousedních oblastech, které sdílí hranu s danou oblastí grafu S. Důvodem vzniku tohoto vylepšení je opět heuristika, která má cíl snížit počet barev v dobrém vrcholovém barvení grafu T. Máme-li síť pokrytou pouze trojúhelníky, pak každá oblast bude mít informaci o třech svých sousedních oblastech, se kterými sousedí přes hranu. Pro lepší vysvětlení uvedeme již známý Obrázek 21.



Obrázek 21: Sestavení grafu T pomocí operace η

Například pro oblast f_1 z Obrázku 21, si budeme uchovávat pouze tři sousední oblasti v novém poli *DominantNeighbors* a to sice f_{12},f_4 a f_8. Pro oblast f_{12} budou sousední oblasti v poli *DominantNeighbors* f_1, f_{11} a f_{13}. Dále předpokládáme, že každá oblast v grafu S má alespoň jednu sousední oblast, se kterou sdílí hranu, a celý graf S je navíc souvislý. Pak pole *DominantNeighbors* využijeme při určení priority během obarvování vrcholů v grafu T. Pole *DominantNeighbors* se využije při vkládání indexů vrcholů do fronty *processQueue*. Tím, že se využije pole *DominantNeighbors* se v barvení budou prioritně barvit kliky velikosti čtyř v grafu T. Algoritmus 14 využívající strukturu *DominantNeighbors* vychází z Algoritmu 13 ze strany 81. Jediným rozdílem je 15. až 20. řádek Algoritmu 14. Jak již bylo zmíněno, do fronty *processQueue* se již nevkládají indexy vrcholů na okolí konkretního vrcholu grafu T, ale pouze nenavštívené vrcholy uložené v poli *DominantNeighbors*. Cyklus na 14. až 19. řádku se provede pro každý zpracovaný vrchol nejvýše třikrát, pro síť složenou pouze z trojúhelníků. Časová složitost je stejná jako v Algoritmu 13, tj. $\mathcal{O}(F)$. Paměťová složitost je také stejná jako u Algoritmu 13, tj. $2F \in \mathcal{O}(F)$.

Algorithm 14 Využití struktury grafu v dobrém vrcholovém barvení grafu T

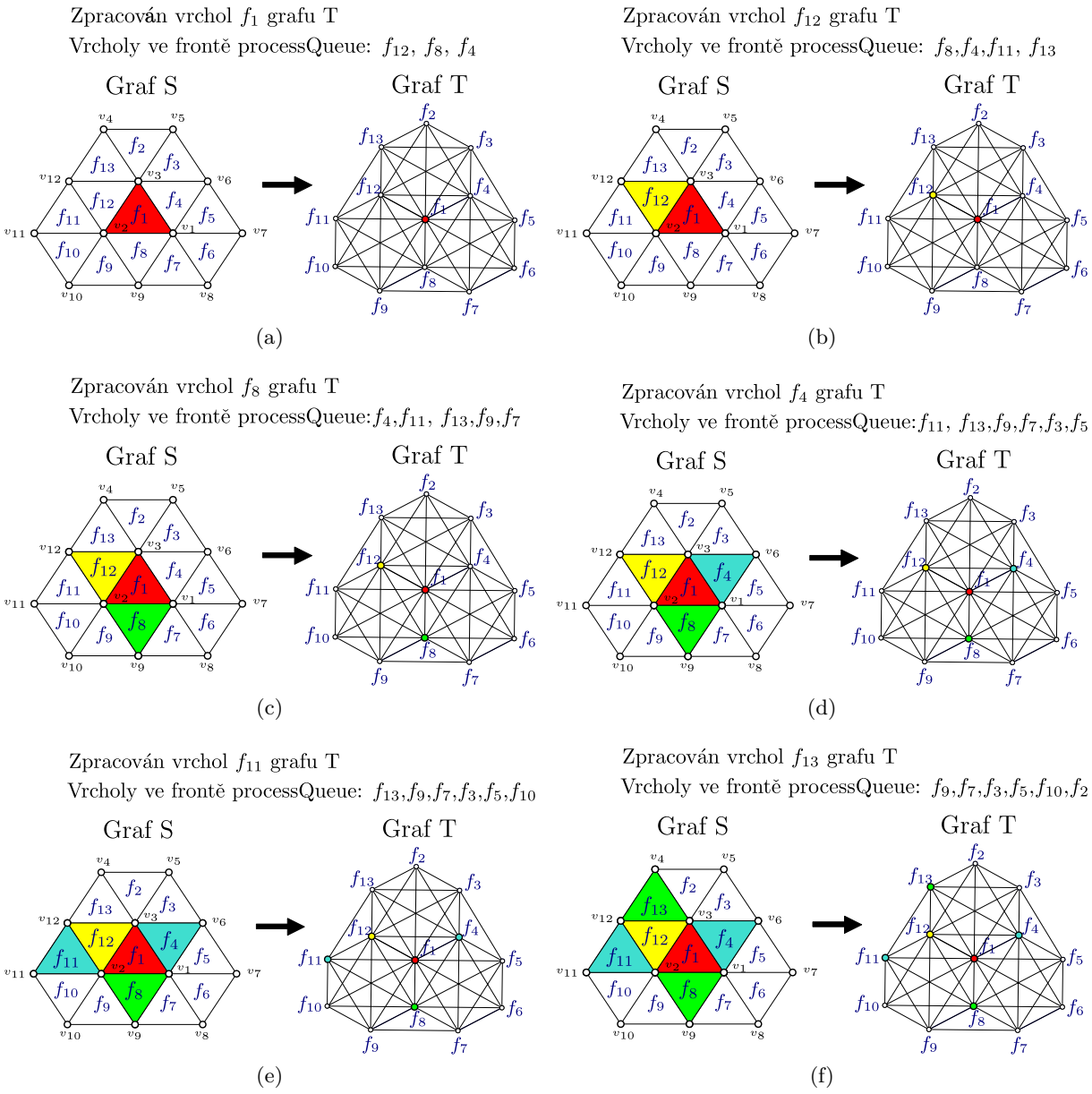
Require: ArraySet ElementNeighbors, start element τ_1, count of elements F, DominantNeighbors

```
 1: maxColorValue ← 6 #  inicializace  ω(T) ≥ 6
 2: ColorElements ← ZeroArray[F]
 3: usedColors ← BitMasking()
 4: State ← ZeroArray[F] # default state 0 - unvisited vertex
 5: processQueue ← Queue()
 6: processQueue.push(τ1)
 7: State[τ1] ← 1 # state 1 - waiting for visit
 8: while !processQueue.empty() do
 9:     τi ← processQueue.pop()
10:     usedColors.clear()
11:     for all τj ∈ ElementNeighbors[τi] do
12:         usedColors.setBit(ColorElements[τj])
13:     end for
14:     for all τj ∈ DominantNeighbors[τi] do
15:         if State[τj] == 0 then
16:             State[τj] ← 1
17:             processQueue.push(τj)
18:         end if
19:     end for
20:     ColorElements[τi] ← usedColors.findFirstFalse()
21:     maxColorValue = max(ColorElements[τi], maxColorValue)
22:     State[τi] ← 2 # state 2 - visited vertex
23: end while
```

Příklad 15

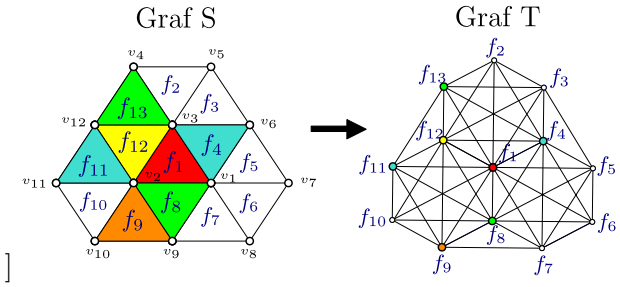
Pro názornost bude vstupem Algoritmu 14 graf T z Obrázku 21. Prvním zpracovaným vrcholem

grafu T v Algoritmu 14 bude vrchol f_1. Další vrcholy grafu T, které se zpracují budou f_{12}, f_8 a f_4. Tím to způsobem se obarví klika velikosti čtyři $\{f_1, f_{12}, f_8, f_4\}$ grafu T, viz Obrázek 21. Další vrcholy, které se přidají do fronty *processQueue* za zpracovaný vrchol f_{12} jsou vrcholy f_{11} a f_{13}. Během zpracovávání a obarvení vrcholu f_8, se do fronty *processQueue* přidají vrcholy f_9 a f_7. Stejným způsobem se za zpracovaný vrchol f_4 přidají do fronty *processQueue* vrcholy f_3 a f_5. Celý běh obarvení grafu T a souvislost s grafem S při použití heuristiky v Algoritmu 13 je znázorněn na Obrázcích 22 a 23. Můžeme vidět, že Algoritmus 14 použil šest barev k dobrému obarvení grafu T, což je dokonce i optimální počet barev, neboť $\omega(T) \geq 6$.



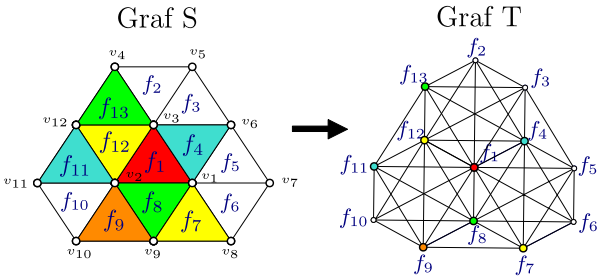
Obrázek 22: Heuristické barvení grafu T - I

Zpracován vrchol f_9 grafu T
Vrcholy ve frontě processQueue: f_7,f_3,f_5,f_{10},f_2



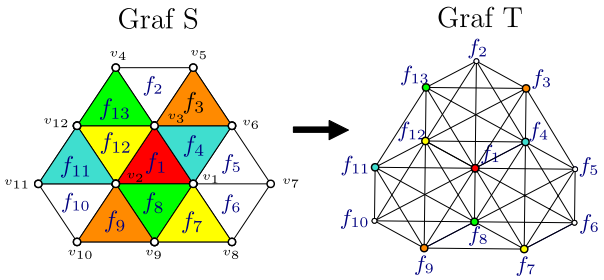
(a)

Zpracován vrchol f_7 grafu T
Vrcholy ve frontě processQueue: f_3,f_5,f_{10},f_2,f_6



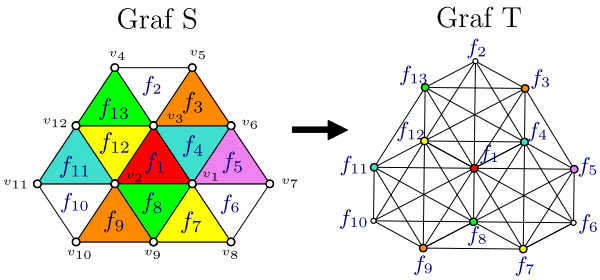
(b)

Zpracován vrchol f_3 grafu T
Vrcholy ve frontě processQueue:f_5,f_{10},f_2,f_6



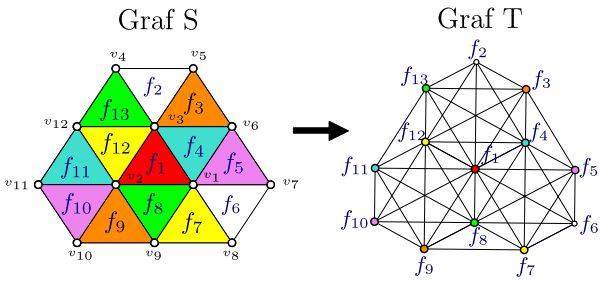
(c)

Zpracován vrchol f_5 grafu T
Vrcholy ve frontě processQueue: f_{10},f_2,f_6



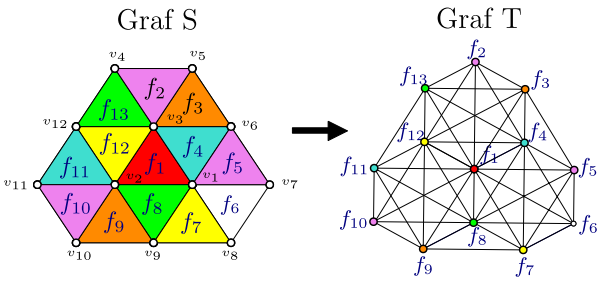
(d)

Zpracován vrchol f_{10} grafu T
Vrcholy ve frontě processQueue: f_2,f_6



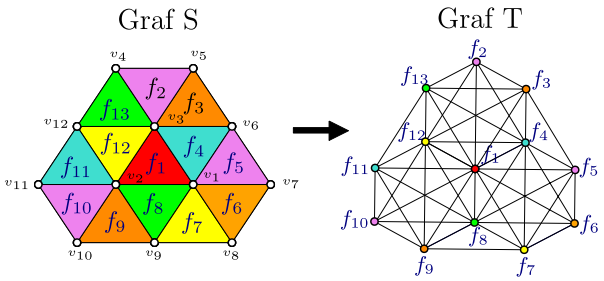
(e)

Zpracován vrchol f_2 grafu T
Vrcholy ve frontě processQueue: f_6



(f)

Zpracován vrchol f_6 grafu T
Vrcholy ve frontě processQueue: $\emptyset$



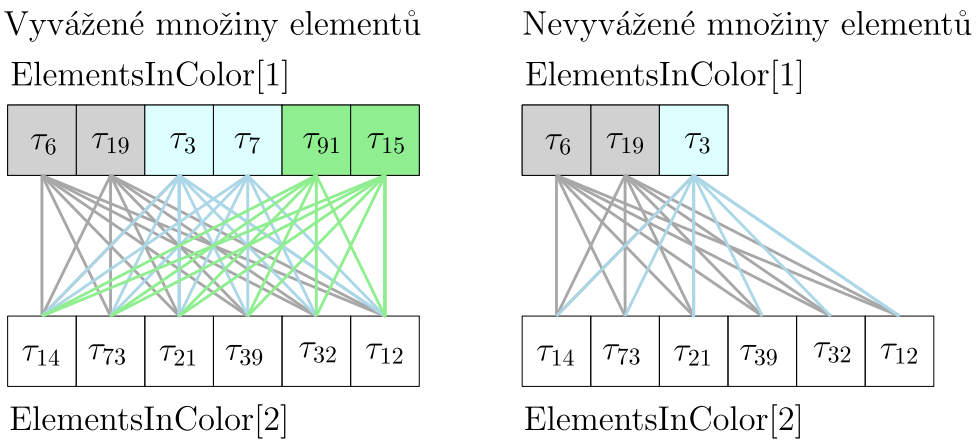
(g)

Obrázek 23: Heuristické barvení grafu T - II

6.3 Rovnoměrné vyvážení počtu vrcholů v barevných třídách

Dalším typem vylepšení je rovnoměrné vyvážení počtů vrcholů v barevných třídách grafu T. Samotné vyvážení má vliv pak na vyvážení počtu vrcholů v grafu H, neboť při vzniku jedné barevné třídy grafu H, je potřeba provést kartezský součin dvou barevných tříd grafu T. Hlavní důvod vziku tohoto vylepšení je efektivní využití paralelismu při sestavení matice $\mathbb{V}$. Podíváme-li se na Algoritmus 11 ze strany 76, vidíme na 8. řádku, že se paralelizuje výpočet nad jedním polem *ElementsInColor*, které symbolizuje jednu barevnou třídu grafu T. Pro lepší představu fungování paralelismu si uveďme malý příklad.

Příklad 16
Představme si dva příklady, ve kterých sestavujeme matici $\mathbb{V}$ v Algoritmu 11. Zaměřme se na to, že v paralelizujeme for smyčku nad polem *ElementsInColor*[1]. Předpokládejme, že na 7. řádků Algoritmu 11 je klauzule *#pragma paralell for schedule(2, dynamic)*. V našem případě to znamená, že velikost každé úlohy, kterou každé vlákno bude počítát jsou dva prvky v poli *ElementsInColor*[1] krát velikost pole *ElementsInColor*[2]. V Obrázku 24 pomocí barev značíme buňky, které zpracovávají jednotlivá vlákna a barevnými spojnicemi jsou označené úlohy, které každé vlákno musí spočítat v příslušné barvě. Pokud bychom předpokládali existenci 3 vláken, tak v nevyváženém případě jsou použité pouze dvě vlákna a třetí vlákno není využité vůbec. Což snižuje efektivitu paralelismu v algoritmu, neboť nejsou paralelní prostředky využity efektivně. Ve vyváženém případě vidíme, že za stejnou dobu se spočítalo 36 úloh za předpokladu že všechny úlohy trvaly výpočetně stejně dlouho. V nevyváženém způsobu vidíme, že se spočítalo pouze 18 úloh, což je polovina. Proto se budeme v této sekci snažit vyvážit počty vrholů v barevných třídách grafu T, které jsou uloženy v poli *ElementsInColor*



Obrázek 24: Ukázka paralelismu při vyvážení počtu vrcholů v barevných třídách grafu T

Zmiňme, že při barvení vrcholů hladovým algoritmem jsou upřednostňovány barvy s nízkou hodnotou. Tento způsob obarvování má za následek větší počet vrcholů v barevných třídách, kde má

barva malé číslo. Je důležité si uvědomit, že takto zvolená heuristika pomocí hladového algoritmu se snaží minimalizovat počet použitých barev tím, že vždy použije nejmenší možnou barvu. Zabránit nerovnoměrnému počtu vrcholů v barevných třídách se budeme snažit dosáhnout v této kapitole. Jedno z řešení, které uvedeme v této kapitole, bude založeno na přebarvování vrcholů. K tomuto postupu je nutné mít dobře vrcholově obarvený graf T. U takto dobře obarveného grafu T můžeme vrcholy z barevných tříd, kde je vrcholů nadbytek, zkusit přebarvit na barvu, kde jich je výrazně méně. Důležité je, aby se počet použitých barev nezvětšil. Popsanou myšlenku s přebarvením můžeme shrnout v následující Algoritmus 15, který je uveden a straně 88. Připomeňme, že Algoritmus 15 pracuje s již obarveným grafem T.

6.3.1 Popis vyvažovacího Algoritmu 15

Předpokladem Algoritmu 15 je dobře vrcholově obarvený graf T. Ve stručnosti popíšeme hlavní myšlenku tohoto vyvažovacího Algoritmu 15. Na začátku Algoritmu 15 je potřeba vypočítat průměrnou hodnotu počtu vrcholů v jedné barevné třídě, která je dána jednoduchým výpočtem $\lceil \frac{F}{maxColor} \rceil$, kde F je počet oblastí a v proměnné $maxColor$ je uložená maximální hodnota barvy, kterou je graf T dobře obarven. Tuto hodnotu v Algoritmu 15 značíme *averageVerticesInColorClass*. Dále je potřeba si uložit výchozí stav o počtech vrcholů v jednotlivých barevných třídách grafu T, tj. pole *colorClassSize*. Při sestavování pole *colorClassSize* můžeme informace o barevných třídách, které mají počet vrcholů, větší než je průměrná hodnota počtů vrcholů v jedné barevné třídě grafu T, uložit do množiny *overfullColorClass*. Tuto množinu *overfullColorClass* využijeme v pozdější fázi, neboť nám bude dávat informaci o tom, z jakých barevných tříd se máme pokoušet o přebarvení vrcholů. Při přebarvování vrcholů snižujeme počet vrcholů v původní barevné třídě a přidáváme vrcholy do tříd, kde jich je méně než je průměrný počet. Když již máme vše nezbytné nachystáno, můžeme dojít k hlavní myšlence Algoritmu 15. Ta je postavena na tom, že Algoritmus 15 prochází jednotlivé barvy, které mají počet vrcholů v barevných třídách větší než je průměrný počet počte vrcholů v jedné barevné třídě. V těchto třídách se snaží heuristicky přebarvit určitý počet vrcholů, dokud se počet vrcholů nerovná průměrnému počtu případně na nejlepší možný počet, který je blízko průměrnému počtu. Opět je i zde je použitý hladový algoritmus, který přiřadí právě přebarvovanému vrcholu buď nejmenší možnou barvu ze třídy, kde je počet vrcholů menší, než je průměrný počet vrcholů, nebo použije novou barvu. Situaci kdy je nutné použít novou barvu Algoritmus 15 nevykoná, neboť nechceme zvětšovat počet použitých barev v dobrém barvení grafu T. Na konci Algoritmu 15 je potřeba znovu sestavit barevné třídy grafu T.

6.3.2 Paměťová a časová složitost vyvažovacího Algoritmu 15

Nyní se podíváme na časovou složitost Algoritmu 15. Sestavení pole *colorClassSize* se provede maximálně $\Delta(T) + 1$ krát, neboť se jedná o horní odhad počtu barev v dobrém barvení grafu T, viz Lemma 2 ze strany 22. Počet barev v množině *overfullColorClass* může být až $\Delta(T)$ v nejhorším případě. Určení počtu operací přebarvení může být složité, zde záleží na topologii grafu T a počtu vrcholů v jednotlivých barevných třídách grafu T. Jako příliš velký horní odhad vezmeme $\Delta(T)F$. V nejhorším případě je potřeba přebarvit celý graf což v sobě obnáší zkontrolovat pro každý vrchol grafu v nejhorším případě $\Delta(T)$ sousedních vrcholů. V poslední fázi sestavujeme nové barevné třídy dané přebarvení s časovou složitostí F. Celková časová složitost Algoritmu 15 je $\Delta(T)+(\Delta(T)+1)F+F = \Delta(T)+(\Delta(T)+2)F \in \mathcal{O}(\Delta(T)F)$. Pro malé hodnoty $\Delta(T)$, lze považovat časovou složitost za $\mathcal{O}(F)$. Je důležité poznamenat, že samotné vyvážení počtu vrcholů v barevných třídách grafu T pomocí Algoritmu 15, nezvýší řádově celkovou časovou složitost barvení grafu T.

Paměťová složitost Algoritmu 15 je $\Delta(T) + 1 + \Delta = 2\Delta(T) + 1 \in \mathcal{O}(1)$. Znovu vytvoření pole *ElementsInColor* považujeme za konstantní, neboť stejnou velikost pole zaujímalo ve vstupu Algoritmu 15.

Algorithm 15 Algoritmus sloužící k rovnoměrnému vyvážení počtu vrcholů v barevných třídách

Require: arraySet ElementNeighbors, element count F, set of elements $\{\tau\}$, array ColorElements, array ElementsInColor, maxColor

```
colorClassSize ← ZeroArray[maxColor]
overfullColorClass ← ∅
averageVerticesInColorClass ← ⌈F/maxColor⌉)
for all color ∈ {1,2,...,maxColor} do
    colorClassSize[color] ← sizeof(ElementsInColor[color])
    if colorClassSize[color] > averageVerticesInColorClass then
        overfullColorClass = overfullColorClass ∪ {color}
    end if
end for
usedColor ← BitMasking()
for all overfullColor ∈ overfullColorClass do
    for all τ_i ∈ ElementsInColor[overfullColor] do
        if colorClassSize[colorOverfull] ≤ averageVerticesInColorClass then
            break
        end if
                                                    ▷ Recoloring element
        usedColor.clear()
        for all τ_j ∈ ElementNeighbors[τ_i] do
            usedColor ← ColorElements[τ_j]
            usedColors.setBit(usedColor)
        end for
        for all color ∈ {1, 2, ..., maxColor} do
            if colorClassSize[color] ≥ averageVerticesInColorClass then
                usedColors.setBit(color)
            end if
        end for
        newColor ← usedColors.findFirstFalse()
        if newColor ≤ maxColor then
            colorClassSize[overfullColor] = colorClassSize[overfullColor] −1
            colorClassSize[newColor] =colorClassSize[newColor] +1
            ColorElements[τ_k] = newColor
        end if
    end for
end for
 ElementsInColor ← ArraySet[maxColor]
for τ_i ∈ {τ} do
    currentColor ← ColorElements[τ_i]
    ElementsInColor[ColorElements[τ_i]] ←ElementsInColor[ColorElements[τ_i]] ∪ {τ_i}
end for
```

6.4 Heuristická redukce počtu použitých barev na dobré vrcholové barvení grafu T

V předchozí sekci 6.3 na straně 85 jsme ukázali řešení vyvážení počtu vrcholů v jednotlivých barevných třídách grafu T. Samotná myšlenka vyvážení počtu vrcholů v barevných třídách grafu T předpokládala dobře obarvený graf T. Následně si vyvažovací Algoritmus 15 vypočítal průměrnou hodnotu počtu vrcholů v jedné barevné třídě grafu T. Na základě této hodnoty se Algoritmus 15 rozhodoval, ze které barevné třídy se bude pokoušet přebarvovat vrcholy na jinou barvu v dobrém vrcholovém barvení grafu T tak, aby neporušil podmínky dobrého barvení grafu T. Nás ovšem může napadnout otázka, co podobnou myšlenku nevyužít na snížení počtu barev v již dobře obarveném grafu T a pokusit se všechny vrcholy z libovolné, ale pevné barevné třídy přebarvit na jiné barvy v dobrém barvení grafu T. Samotné snížení počtu použitých barev v dobrém barvení grafu T má velký dopad na snížení počtů barev v dobrém barvení grafu H.

Příklad 17
Předpokládejme, že se nám podaří graf T dobře obarvit pomocí 9 barev. To pro graf H znamená, že jej umíme dobře obarvit pomocí $9 \cdot 9 = 81$ barev. Pokud by nějaký algoritmus byl schopen snížit počet barev grafu T byť jen o jednu barvu (tedy využít k dobrému vrcholovému barvení grafu T pouze 8 barev), pro dobré vrcholové barvení grafu H by to znamenalo použit jen $8 \cdot 8 = 64$ barev. Jedná se zde o úsporu 17 barev v dobrém vrcholovém barvení grafu H. To má za následek většího počtu vrcholů v barevných třídách, pokud jsou rovnoměrně rozmístěné a využit efektivněji paralelismu, protože může být větší podúloha být zpracována paralelně a rychleji. Odpadá zde 17 čekání mezi vlákny před startem nové podúlohy.

Heuristický Algoritmus 16 bude do jisté míry podobný Algoritmu 15 ze stránky 88. Hlavní myšlenka Algoritmu 16 se týká toho, že na vstupu Algoritmu 16 máme již dobře vrcholově obarvený graf T. Algoritmus 16 vyžaduje penalizační barvu a samotné barevné třídy grafu T. Úkolem Algoritmu 16 je rozhodnout a pokusit se přebarvit všechny vrcholů z jedné barevné třídy, která je určena hodnotou penalizační barvy, na jinou barvu v dobrém barvení grafu T. Během přebarvování vrcholů nesmí Algoritmus 16 porušit podmínku dobrého barvení. Pokud by byla porušená podmínka dobrého barvení grafu T, mohly by vzniknout kolize při zápisu hodnot z lokálních matic $\mathbb{V}_{loc}^{k,l}$ do matice $\mathbb{V}$ během paralelního výpočtu v metodě hraničních prvků. Jestliže Algoritmus 16 skončí úspěšně a vrátí návratovou hodnotu *True*, znamená to, že se mu podařilo snížit počet barev o jedničku v dobrém barvení grafu T, tím že přebarvil všechny vrcholy, které dřív byly obarvené penalizační barvou. V opačném případě Algoritmus 16 vrátí hodnotu *False*, která signalizuje, že není možné všechny vrcholy obarvené penalizační barvou takto jednoduše přebarvit na jiné barvy v dobrém barvení grafu T.

Algorithm 16 Algoritmus sloužící k redukci počtu barev za pomocí přebarvování vrcholů v barevných třídách

Require: arraySet ElementNeighbors, element count F, set of elements $\{\tau\}$, array ColorElements, array ElementsInColor, color maxColor, color penalizeColor

```
1: usedColor ← BitMasking()
2: for all τ_j ∈ ElementsInColor[penalizeColor] do
3:     usedColor.clear()
4:     usedColor.setBit(penalizeColor)
5:     for all  τ_i ∈ ElementNeighbors[τ_j] do
6:         usedColors.setBit(ColorElements[τ_i])
7:     end for
8:     newColor ← usedColors.findFirstFalse()
9:     if newColor ≤ maxColorValue then
10:        ColorElements[τ_j] ← newColor
11:    else
12:         ElementsInColor ← ArraySet[maxColorValue]
13:        for τ_i ∈ {τ} do
14:            currentColor ← ColorElements[elementIndex]
15:            ElementsInColor[ColorElements[τ_i]] = ElementsInColor[ColorElements[τ_i]] ∪{τ_i}
16:        end for
17:        return False
18:    end if
19: end for
20: if penalizeColor ≠ maxColorValue then
21:    for τ_j ∈ {τ}  do
22:        if ColorElements[τ_j] == maxColorValue then
23:            ColorElements[τ_j] = penalizeColor
24:        end if
25:    end for
26: end if
27: maxColorValue = maxColorValue−1
28:  ElementsInColor ← ArraySet[maxColorValue]
29: for τ_i ∈ {τ} do
30:    currentColor ← ColorElements[elementIndex]
31:    ElementsInColor[ColorElements[τ_i]] = ElementsInColor[ColorElements[τ_i]] ∪{τ_i}
32: end for
33: return True
```

6.4.1 Popis Algoritmu 16

Jak již bylo zmíněno v odstavci výše, Algoritmus 16 pracuje s dobře vrcholově obarveným grafem T. Proto opět očekává na vstupu pole *ColorElements*, které v sobě obsahuje informaci o přiřazené barvě pro každý vrchol v grafu T. Dále se zde v proměnné *penalizeColor* nachází celočíselná hodnota penalizační barvy. Vrcholy obarvené penalizační barvou se Algoritmus 16 pokusí přebarvit na jinou barvu v dobrém vrcholovém barvení grafu T. S tím souvisí i potřeba

vyžadovat na vstupu Algoritmu 16 barevné třídy grafu T, které jsou uložené v poli *ColorClass*, proto, aby bylo snazší procházet a přebarvovat pouze vrcholy grafu T, které jsou obarvené penalizační barvou. Na vstupu Algoritmu 16 se vyskytuje pole *ElementNeighbors*, které v sobě uchovává pro všechny vrcholy grafu T množinu sousedních vrcholů. Pole uložené v proměnné *ElementNeighbours* Algoritmus 16 využije ke zjištění barvy sousedních vrcholů tak, aby při přebarvování vrcholů v grafu T, byla vždy splněna podmínka dobrého vrcholového barvení. Opět i zde využijeme ke zjištění volné barvy hladový algoritmus, který testuje barvy na okolí každého vrcholu z barevné třídy určené penalizační barvou. Pokud se Algoritmu 16 podaří přebarvit všechny vrcholy obarvené penalzační barvou, vrátí Algoritmus 16 hodnotu *True* a znovu sestaví barevné třídy grafu T v poli *ElementsInColor*. Při neúspěšném přebarvení vrátí hodnotu *False* a opět sestaví barevné třídy grafu T.

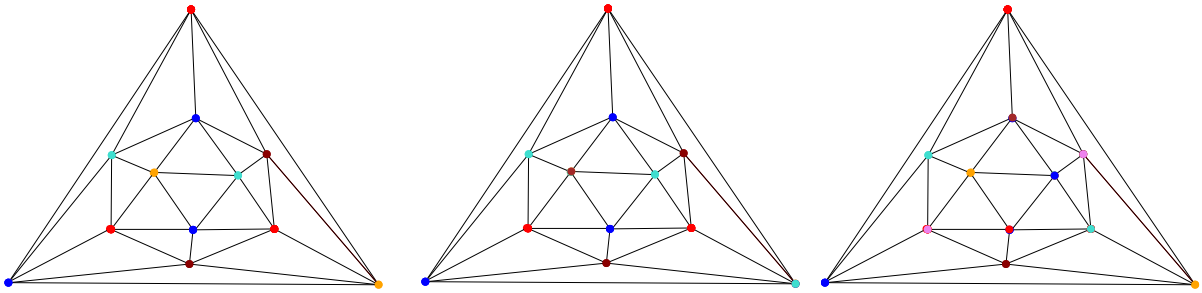
6.4.2 Časová a paměťová náročnost Algoritmu 16

Nyní se pokusíme určit časovou složitost Algoritmu 16. Víme že toto určení hodně závisí na počtu vrcholů v jednotlivých barevných třídách. V nehorším případě bude potřeba přebarvit téměř všechny vrcholy v grafu T. Nejhorší případ by mohl vypadat tak, že by v jedné barevné třídě byly zastoupeny skoro všechny vrcholy grafu T a v ostatních třídách by byly vrcholy zastoupeny pouze v jednotkách. V tom případě můžeme odhadnout časovou složitost na $\Delta(T)F = 12F$. Navíc při úspěšném i při neúspěšném přebarvení, je potřeba vytvořit nové barevné třídy grafu T. To má časovou složitost F. Pokud sečteme všechny časové složitosti, obdržíme celkovou časovou složitost, která je rovna $\Delta(T)F + F = (\Delta(T) + 1)F \in \mathcal{O}(\Delta(T)F)$. Pro malé hodnoty $\Delta(T)$, můžeme časovou složitost Algoritmu 16 považovat za $\mathcal{O}(F)$.

Protože v tomto Algoritmu 16 nevytváříme žádné pole, kromě znovu vytvoření barevných tříd grafu T v poli *ElementsInColor*. Protože však nové pole *ElementsInColor* má stejnou velikost jako vstupní pole nepovažujeme tento krok za paměťovou složitost. Proto paměťová složitost Algoritmu 16 je $\mathcal{O}(1)$.

Příklad 18
Pro lepší pochopení samotného Algoritmu 16 uvádíme na Obrázku 25 příklady tří grafů pravidelného dvacetistěnu. Vstupním grafem v Algoritmu 16 bude graf A z Obrázku 25, který je dobře vrcholově obarven pomocí pěti barev. Pokud bychom vybrali jako penalizační barvu modrou nebo červenou barvu, tak po vykonání Algoritmu 16 obdržíme totožně obarvený graf A, neboť všechny vrcholy modré a červené barvy mají na svém okolí barvy všech zbylých čtyřech barev a není proto možné žádný vrchol přebarvit na jinou barvu. Pokud ovšem zvolíme jako penalizační barvu oranžovou barvu, dostaneme po vykonání Algoritmu 16 obarvený graf B z Obrázku 25,

který již obsahuje čtyři barvy. Pokud jako vstup Algoritmu 16 zvolíme graf C z Obrázku 25, zjistíme, že ať zvolíme jakoukoliv penalizační barvu, vždy obdržíme totožný graf C dobře vrcholově obarvený 6 barvami. To znamená, že ne vždy se podaří Algoritmu 16 zmenšit počet barev v dobrém vrcholovém barvení.



Graf A Graf B Graf C

Obrázek 25: Dobře vrcholově obarvené grafy pravidelného dvacetistěnu

Důvodem proč se v této práci zaměřujeme na heuristické algoritmy je ten, že v současné době mají deterministické algoritmy dobrého vrcholového barvení exponenciální třídu časové náročnosti. To se v praxi ukazuje jako neefektivní použití. Z tohoto důvodu se zaměřujeme na heuristické algoritmy, které mají polynomiální někdy i lineární časovou náročnost. Pro svou časovou náročnost jsou heuristické algoritmy rychlé a hojně využívané v praktických aplikacích.

6.5 Shrnutí

V této kapitole jsme předvedli několik druhů vylepšení. Nejdříve jsme ukázali využití efektivní struktury *BitMasking*, která využívá bitové reprezentaci čísel pro zjišťování nepoužité barvy na okolí vrcholů. Toto vylepšení má snížit výpočetní čas strávený během barvení grafu T. Dále jsme uvedli dva heuristické algoritmy barvení grafu T, které ke svému obarvení využívaly strukturu grafu T. Úkolem heuristických algoritmů je využít co nejméně barev na dobré vrcholové obarvení grafu T. V dalším příkladu jsme se zabývali rovnoměrným vyvážením počtu barev v barevných třídách grafu T. Důvod vzniku tohoto vylepšení je, aby velikosti podúloh byly stejné. Tím zamezíme tomu, že v některé barevné třídě grafu T bylo málo vrcholů. To by mohlo způsobit během vykonávání paralelní smyčky přes *color*, tj. barevnou třídu T stav, ve kterém by některá vlákna byla nevyužita. Proto chceme využít paralelismus efektivně a rozdělujeme úlohy do rovnoměrných tříd. Posledním vylepšením, které jsme uvedli, je heuristický Algoritmus 16, který se pokouší o snížení počtu barev v grafu T. Připomeňme si, že počet použitých barev na dobré vrcholové barvení grafu T má kvadratický vliv na dobré vrcholové barvení grafu H, proto úspora může být významaná.

7 Numerické experimenty

V této kapitole opět budeme předpokládat, že v metodě hraničních prvků spolu interagují po částech lineární bázové funkce. Kapitola 7 je rozdělena na tři hlavní sekce. V první sekci se zaměříme na efektivitu námi navrženého postupu řešení během výpočtu matice $\mathbb{V}$ v metodě hraničních prvků (za použití dobrého vrcholového barvení grafu H). Připomeňme, že k samotnému sestavení barvení grafu H využijeme dobře vrcholově obarvený graf T. Časové výsledky budeme porovnávat s již implementovanou podporou ošetřené kritické sekce za pomocí atomické operace v knihovně BEM4I [11], které budeme nadále značit jako **AlgReferenceAtomic**. Veškeré testy budeme provádět na výpočetním clusteru Barbora [6], který je provozován IT4Innovations národním superpočítačovým centrem. Za poskytnuté hardwarové prostředky tímto velice děkuji. V druhé sekci se budeme zabývat barvením testovacích grafů T, které postupně vzniknou z testovacích sítí a grafů S. U testovacích barvících algoritmů nás budou zajímat tyto veličiny:

- Počet oblastí tj. počet trojúhelníku dané sítě
- Čas sestavení grafu T
- Čas dobrého vrcholového barvení grafu T
- Maximální počet použitých barev na dobré vrcholové barvení grafu T

Taktéž se zaměříme na vyvažovací Algoritmus 15 popsaný na straně 88. K testování využijeme knihovnu BEM4I [11]. Za testovací sítě objektů využijeme již předpřipravené sítě z knihovny BEM4I [11], které se nachází ve složce *inputs*. Ve třetí sekci shrneme námi dosažené výsledky.

Superpočítač Barbora [6]

Všechny numerické testy jsme prováděli na superpočítači Barbora [6]. Všechny výsledky byly vykonávány na jednom uzlu počítače Barbora. Tento výpočetní uzel superpočítače Barbora byl osazen jedním 64 bitovým, 18 jádrovým procesorem Intel Cascade Lake 6240. Pracovní frekvence tohoto procesoru je 2.6 GHz. Tato pracovní frekvence procesoru se může zvýšit až na 3.9 GHz. Procesor disponuje 36 paralelními vlákny, které jsme všechny využili při řešení Laplaceovy úlohy při testech škálovatelnosti.

Knihovna BEM4I [11]

Knihovna BEM4I slouží k numerickému výpočtu metody hraničních prvků. Je napsaná v jazyce C++ ve verzi C++11. Knihovna je vyvíjená týmem z Národního superpočítačového centra IT4Innovations. Knihovna BEM4I [11] využívá k paralelnímu vykonávání knihovnu OpenMP ve

verzi 5.0. Knihovna OpenMP pracuje na principu tzv. fork join modelu. Všechny námi prováděné numerické výsledky byly překládány s nejnovější instrukční sadou *-avx512*, která je podporována procesorem Intel Cascade Lake 6240.

7.1 Řešení metody hraničních prvků

V této sekci uvedeme dosažené výsledky pomoci námi navrženého barvení grafu H. Barvení grafu H při sestavení matice $\mathbb{V}$během řešení Laplaceovy úlohy pomocí metody hraničních prvků v knihovně BEM4I [11]. Během řešení metody hraničních prvků, byly použity po částech lineární bázové funkce. Výsledky porovnáme s již implementovaným řešením, ve kterém jsou kolize ošetřené pomocí atomická operace. Existující implementované řešení v knihovně BEM4I [11] budeme nadále v textu značit **AlgReferenceAtomic**. Během testování nás bude zajímat:

- Čas sestavení matic $\mathbb{V}$ a $\mathbb{K}$
- Absolutní a relativní chyba vypočteného řešení
- Parametry vyvážení úlohy během paralelního zpracování
- Čas samotného barvení grafu T
- Počet použitých barev v dobrém vrcholovém barvení grafu H

K testování využijeme Algoritmus 12, který pojmenujeme **AlgColoringH**. Algoritmus **AlgColoringH** bude využívat barvícího Algoritmu 14 ze strany 82, kde se v grafu T barví kliky o velikosti čtyř. Algoritmus **AlgColoringH** využije vyvažovacího Algoritmu 15 ze strany 88. Jednotlivé naměřené časy sestavení matic $\mathbb{V}$ a $\mathbb{K}$ budeme zaznamenávat do Tabulky 21. K testování využijeme modely sítí, které se nachází ve složce *inputs* v knihovně BEM4I [11] (icosahedron.txt, cube_24.txt). Námi navržené řešení se nachází ve větvi *feature/coloring* verzovacího systému Git [4] umístěného v knihovně BEM4I [11]. Naše revizní větev *feature/coloring* vychází z verze knihovny, který je označen *commitem* s otiskem *0c21742cf0a42ee2ba440ed318f537673bff2db5*.

7.1.1 Průběh testování

V Tabulce 21 jsme zaznamenali časy sestavení matic $\mathbb{V}$ a $\mathbb{K}$, které jsme testovali na čtyřech různých sítích. Provedli jsme vždy šest testů, kde první naměřený výsledek testů jsme zahodili a ze zbylých pěti testů jsme udělali aritmetický průměr a zaznamenali do příslušné tabulky. Během testování jsme sledovali absolutní a relativní chybu řešení, která byla stejná s referenčním řešením napříč všemi výpočty. To je dobře, neboť máme jistotu, že námi implementované řešení opravdu řeší problém s kritickou sekcí a nevznikají nekonzistentní data v globální matici $\mathbb{V}$. Poznamenejme, že metodu sestavení a obarvení grafu T jsme využili při sestavení matice $\mathbb{V}$. V čase sestavení matice $\mathbb{V}$ je zahrnutý i čas sestavení a barvení grafu T. Jednotlivé časy sestavení a obarvení grafu T jsou podrobně popsány v následující sekci 7.3 na straně 102. Během testování jsme vyzkoušeli celkem 8 testů, včetně testu referenčního řešení **AlgReferenceAtomic**. Jediným parametrem, který jsme v prostředí paralelní knihovny OpenMP [12] měnili, byl způsob zpracování výpočtů interakce bázových funkcí nad dvojicemi elementů, a s tím spojená velikost podúlohy, tj. počet interakcí dvojic, které, kterou každé vlákno během paralelního běhu zpracovalo. Na výběr jsme měli celkem ze čtyř možností plánování: *static*, *dynamic*, *guided* a *auto*. Plánování *static*, *dynamic* a *guided* mohou obsahovat číselný parametr velikosti podúlohy, který připadne každému vláknu. Jednotlivé druhy plánování si nyní představíme.

- Plánování *static* - Výpočetní úloha je rozdělena na stejně veliké podúlohy fixní velikosti. Plánovač rozdělí podúlohy vláknům před samotným vykonávání úlohy v tzv. kruhovém pořadí.
- Plánování *dynamic* - Výpočetní úloha je taktéž rozdělená na stejně veliké podúlohy fixní velikosti. Vlákna žádají plánovač o další podúlohy, dokud není celá úloha zpracována.
- Plánování *guided* - Výpočetní úloha je taktéž rozdělená na podúlohy. Vlákna žádají plánovač o další podúlohy, dokud není celá úloha zpracována. Velikost podúlohy, kterou vláknům plánovač přidělí, je zdola omezená parametrem velikostí úlohy. V posledním běhu paralelních vláken může být podúloha menší, než je hodnota parametru velikosti podúlohy. O velikosti zpracované podúlohy si plánovač rozhodne sám.
- Plánování *auto* - Plánovač si sám rozhodne, jaký z výše popsaných plánování *static*, *dynamic*, *guided* zvolí.

Protože se paralelizovala vnější smyčka, tj. počet vrcholů obarvené *color*, je počet dvojic, které každé vlákno vykoná násobkem parametru úlohy s počtem vrcholů v barevné třídě obarvené *odstínovou barvou*. Kvůli vyváženém počtu interakcí v jednotlivých vláknech jsme se snažili, aby počty vrcholů v jednotlivých barevných třídách byly zastoupeny rovnoměrně. Z Tabulky 21 vidíme, že se nám nepodařilo zrychlit a překonat současné řešení **AlgReferenceAtomic**, které řeší kolize označením problémové sekce v kódu jako atomickou operaci. Místo toho se nám poda-

řilo časově vyrovnat s referenčním řešením. Časové rozdíly jsou v jednotkách procent. Nejlepší námi navržené řešení bylo opatřit vnější nezávislou paralelní smyčku parametry (*guided*, 1), případně (*auto*). Výběr samotného algoritmu **AlgColoringH** předcházely podmínky např. minimální počet barev v dobrém vrcholovém barvení grafu T. Počet barev v dobrém vrcholovém barvení grafu T má kvadratický vliv na počet barev v dobrém vrcholovém barvení grafu H.

Připomeňme, že v paralelním běhu metody hraničních prvků při sestavení matice $\mathbb{V}$, jsou paralelní výpočty vykonávány nad dvojicemi elementů, tj. vrcholy grafu H, které jsou obarvené stejnou barvou. Samotné zhodnocení a sestavení dobrého vrcholového barvení grafu T je popsáno v sekci 7.3 na straně 102. Při výběru testovacích parametrů jsme dali přednost plánování *dynamic*, neboť nepředpokládáme, že výpočet interakce bázových funkcí nad dvojicemi elementů bude vždy stejně časově náročný. Ne všechny stejně veliké podúlohy musí trvat stejně dlouho. Promítá se zde například vzdálenost buněk v globální matici, kde se zapisuji mezivýsledky interakce bázových funkcí. Proto se nabízí použit plánování *dynamic* oproti plánování *static*. Na odhad parametru velikosti podúlohy žádná univerzální rada neexistuje. Jednou z možností je provést test škálovatelnosti, tak jsme ho provedli v Tabulce 21.

Z Tabulky 21 je vidět, že v parametru *dynamic* při klesající velikosti podúlohy, narůstá čas sestavení matice $\mathbb{V}$. V Tabulce 21 vidíme, že parametr 36 u plánování *dynamic* způsobil zhoršení času sestavení matice $\mathbb{V}$, oproti možnosti *dynamic* 16. Ve všech testech, které jsou zaznamenané v Tabulce 21, našel Algoritmus **AlgColoringH** vždy dobré obarvení grafu T pomocí devíti barev. To znamená, že na dobré obarvení grafu H použil 81 barev. Dále z testu vyplývá, že do 20000 elementů sítě jsou algoritmy **AlgReferenceAtomic** a **AlgColoringH** s parametrem *auto* téměř shodné při sestavení matice $\mathbb{V}$. Čas sestavení a obarvení grafu T je zanedbatelný vzhledem k času výpočtu matice $\mathbb{V}$. Tento argument vznikl v sekci 7.3 na straně 102. Pro zvýšení efektivity jsme části kódu, které se opakují, tj. opakující se dotazování souřadnic elementů v globální matici uložili do dynamického pole. I přes to se nepodařilo dosáhnout zrychlení paralelního výpočtu matice $\mathbb{V}$ v metodě hraničních prvků za použití barvení grafu H na vícejádrové architektuře.

Algoritmus	Čas sestavení matic $\mathbb{V}$ a $\mathbb{K}$	Absolutní a relativní chyba	Parametry
Model koule pokrytý síti o 10242 uzlech, 20480 elementech a 30720 hranách			
AlgReferenceAtomic	$\mathbb{V}$: 12.16 s, $\mathbb{K}$: 13.20 s	Abs: 5.80e-05, Rel: 3.90e-06	dynamic, 8
AlgColoringH	$\mathbb{V}$: 13.67 s, $\mathbb{K}$: 13.16 s	Abs: 5.80e-05, Rel: 3.90e-06	dynamic, 36
AlgColoringH	$\mathbb{V}$: 12.37 s, $\mathbb{K}$: 13.19 s	Abs: 5.80e-05, Rel: 3.90e-06	dynamic, 16
AlgColoringH	$\mathbb{V}$: 12.54 s, $\mathbb{K}$: 13.25 s	Abs: 5.80e-05, Rel: 3.90e-06	dynamic, 8
AlgColoringH	$\mathbb{V}$: 12.56 s, $\mathbb{K}$: 13.20 s	Abs: 5.80e-05, Rel: 3.90e-06	dynamic, 4
AlgColoringH	$\mathbb{V}$: 12.87 s, $\mathbb{K}$: 13.19 s	Abs: 5.79e-05, Rel: 3.90e-06	dynamic, 1
AlgColoringH	$\mathbb{V}$: 12.35 s, $\mathbb{K}$: 13.20 s	Abs: 5.79e-05, Rel: 3.90e-06	guided, 1
AlgColoringH	$\mathbb{V}$: 12.24 s, $\mathbb{K}$: 13.18 s	Abs: 5.79e-05, Rel: 3.90e-06	auto
Model koule pokrytý síti o 40962 uzlech, 81920 elementech a 122880 hranách			
AlgReferenceAtomic	$\mathbb{V}$: 192.94 s, $\mathbb{K}$: 213.33 s	Abs: 5.79e-06, Rel: 3.90e-07	dynamic, 8
AlgColoringH	$\mathbb{V}$: 216.33 s, $\mathbb{K}$: 212.57 s	Abs: 5.79e-06, Rel: 3.90e-07	dynamic, 36
AlgColoringH	$\mathbb{V}$: 200.13 s, $\mathbb{K}$: 213.14 s	Abs: 5.79e-06, Rel: 3.90e-07	dynamic, 16
AlgColoringH	$\mathbb{V}$: 200.18 s, $\mathbb{K}$: 212.72 s	Abs: 5.79e-06, Rel: 3.90e-07	dynamic, 8
AlgColoringH	$\mathbb{V}$: 200.05 s, $\mathbb{K}$: 212.78 s	Abs: 5.79e-06, Rel: 3.90e-07	dynamic, 4
AlgColoringH	$\mathbb{V}$: 201.82 s, $\mathbb{K}$: 212.29 s	Abs: 5.79e-06, Rel: 3.90e-07	dynamic, 1
AlgColoringH	$\mathbb{V}$: 197.19 s, $\mathbb{K}$: 212.58 s	Abs: 5.79e-06, Rel: 3.90e-07	guided, 1
AlgColoringH	$\mathbb{V}$: 196.35 s, $\mathbb{K}$: 213.25 s	Abs: 5.79e-06, Rel: 3.90e-07	auto
Algoritmus	Čas sestavení matic $\mathbb{V}$ a $\mathbb{K}$	Absolutní a relativní chyba	Parametry
Model krychle pokrytý síti o 12290 uzlech, 24576 elementech a 36864 hranách			
AlgReferenceAtomic	$\mathbb{V}$: 17.50 s, $\mathbb{K}$: 19.29 s	Abs: 1.96e-04, Rel: 1.32e-05	dynamic, 8
AlgColoringH	$\mathbb{V}$: 24.10 s, $\mathbb{K}$: 19.23 s	Abs: 1.96e-04, Rel: 1.32e-05	dynamic, 36
AlgColoringH	$\mathbb{V}$: 18.46 s, $\mathbb{K}$: 19.19 s	Abs: 1.96e-04, Rel: 1.32e-05	dynamic, 16
AlgColoringH	$\mathbb{V}$: 18.56 s, $\mathbb{K}$: 19.26 s	Abs: 1.96e-04, Rel: 1.32e-05	dynamic, 8
AlgColoringH	$\mathbb{V}$: 18.56 s, $\mathbb{K}$: 19.27 s	Abs: 1.96e-04, Rel: 1.32e-05	dynamic, 4
AlgColoringH	$\mathbb{V}$: 18.47 s, $\mathbb{K}$: 19.23 s	Abs: 1.96e-04, Rel: 1.32e-05	dynamic, 1
AlgColoringH	$\mathbb{V}$: 17.70 s, $\mathbb{K}$: 19.21 s	Abs: 1.96e-04, Rel: 1.32e-05	guided, 1
AlgColoringH	$\mathbb{V}$: 17.62 s, $\mathbb{K}$: 19.22 s	Abs: 1.96e-04, Rel: 1.32e-05	auto
Model krychle pokrytý síti o 49154 uzlech, 98304 elementech a 147456 hranách			
AlgReferenceAtomic	$\mathbb{V}$: 275.13 s, $\mathbb{K}$: 305.95 s	Abs: 1.25e-04, Rel: 8.44e-06	dynamic, 8
AlgColoringH	$\mathbb{V}$: 301.98 s, $\mathbb{K}$: 306.56 s	Abs: 1.25e-04, Rel: 8.44e-06	dynamic, 36
AlgColoringH	$\mathbb{V}$: 286.63 s, $\mathbb{K}$: 304.97 s	Abs: 1.25e-04, Rel: 8.44e-06	dynamic, 16
AlgColoringH	$\mathbb{V}$: 287.15 s, $\mathbb{K}$: 306.64 s	Abs: 1.25e-04, Rel: 8.44e-06	dynamic, 8
AlgColoringH	$\mathbb{V}$: 287.66 s, $\mathbb{K}$: 306.72 s	Abs: 1.25e-04, Rel: 8.44e-06	dynamic, 4
AlgColoringH	$\mathbb{V}$: 291.83 s, $\mathbb{K}$: 304.87 s	Abs: 1.25e-04, Rel: 8.44e-06	dynamic, 1
AlgColoringH	$\mathbb{V}$: 283.33 s, $\mathbb{K}$: 305.58 s	Abs: 1.25e-04, Rel: 8.44e-06	guided, 1
AlgColoringH	$\mathbb{V}$: 282.98 s, $\mathbb{K}$: 306.23 s	Abs: 1.25e-04, Rel: 8.44e-06	auto

Tabulka 21: Testování času sestavení globálních matic $\mathbb{V}$ a $\mathbb{K}$ v knihovně BEM4I [11]

7.2 Zjištění časového zpomalení sestavení matice $\mathbb{V}$ pomocí barvení

Opět i v této sekci budeme řešit Laplaceovou úlohu pomocí metody hraničních prvků v knihovně BEM4I [11]. Bázové funkce budou po částech lineární. Hlavním smyslem této sekce je přijít na příčinu, proč sestavení matice $\mathbb{V}$ pomocí Algoritmu **AlgColoringH** nebylo rychlejší než referenční řešení **AlgReferenceAtomic**. Nyní si představíme pár variant testovacích algoritmů, které budou vycházet z referenčního řešení **AlgReferenceAtomic**. Tyto algoritmy budeme přetvářet na implementaci použití Algoritmu **AlgColoringH** při sestavení matice $\mathbb{V}$.

- **AlgRefAtomVec1** - budeme značit upravený Algoritmus **AlgReferenceAtomic**, který bude přistupovat k prvkům jednodimenzionálního pole *std::vector<int> items*. Pole *items* využijeme pro získání konkrétních indexů elementů. V poli *items* je pod indexem i uložená celočíselná hodnota i, tj. *items[i]* $= i$. Obdobně jako v Algoritmu **AlgColoringH** máme jednotlivé obarvené elementy uložené ve stejné struktuře. Tento test provádíme z důvodu zjištění, zda použití přístupu k prvkům v poli *items* má časový dopad na sestavení matice $\mathbb{V}$.

- **AlgRefAtomVec1Rand** - budeme značit upravený Algoritmus **AlgReferenceAtomic**. Tento Algoritmus **AlgRefAtomVec1Rand** bude téměř stejný s Algoritmem **AlgRefAtomVec1**. Jediný rozdíl je, že pod příslušným indexem pole *items[i]* se nebudou nacházet celočíselná hodnota i, ale hodnota indexu jiného elementu. Indexy v poli budou nesetříděné, obdobně jako obarvené elementy v barevných třídách grafu T neodpovídají setříděné posloupnosti indexů elementů. Tento test provádíme z důvodu, zda má uspořádání prvků v poli *std::vector<int> items* časový dopad na sestavení matice $\mathbb{V}$.

- **AlgRefAtom9Rand** - budeme značit Algoritmus, který bude využívat jednorozměrného pole *std::vector<int> items*. Podobně jako v Algoritmu **AlgRefAtomVec1Rand** budou v poli *items* náhodně uspořádané indexy elementů sítě. Dále v Algoritmu **AlgRefAtom9Rand** využijeme pole *ranges*, ve kterém budou uložené jednotlivé intervaly rovnoměrně rozdělených elementů v poli *items* na 9 části. Podobně jako se nám podařilo pomocí Algoritmu **AlgColoringH** rozdělit množinu elementů na 9 disjunktních množin pomocí dobrého vrcholového barvení grafu T, tj. obarvit graf T pomocí 9 barev. Připomeňme, že vrcholy grafu T jsou elementy sítě.

  Pro příklad mějme síť složenou z 20 elementů. V poli *items* budou uložené jednotlivé indexy elementů, tj. hodnoty $\{0, 1, \ldots, 19\}$ v libovolném pořadí. Dále předpokládejme, že budeme chtít pole *items* rozdělit rovnoměrně na 4 intervaly, tj. $\langle 0; 5), \langle 5; 10), \langle 5; 15), \langle 15; 20)$. Tyto intervaly budeme reprezentovat pomocí pole *range* následovně: *range[0] = 0*, *range[1] = 5*, *range[2] = 10*, *range[3] = 15*, *range[4] = 20*.

  Stejně jako je smyčka přes *color* a *shade* barvu v Algoritmu **AlgColoringH**, tak i v tomto

algoritmu budeme sestavovat matici $\mathbb{V}$ přes imaginární *color* a imaginární *shade* barvu. Jednotlivé imaginární barevné třídy budou uloženy za sebou v poli *items*. Intervaly v poli *ranges* odpovídají intervalům imaginárních barevných tříd v poli *items*. Při výpočtu budou mezi sebou interagovat všechny elementy napříč imaginárními barevnými třídami imaginarního grafu H. Tento test provádíme proto, zda jednotlivé čekání vláken mezi sebou před začátkem nové zpracované imaginární *color* a *shade* barvy má časový dopad na sestavení matice $\mathbb{V}$. Připomeňme, že v tomto testu se bude jednat o $9 \cdot 9 = 81$ čekání, stejně jako v Algoritmu **AlgColoringH**. Hlavním rozdílem oproti Algoritmu **AlgColoringH** je ošetření kolizí pomocí atomické sekce. Očekáváme, že v tomto algoritmu pomalejší čas sestavení matice než je v Algoritmu **AlgColoringH**.

- **AlgRefAtom9Sort** - budeme značit Algoritmus, který bude téměř totožný s Algoritmem **AlgRefAtom9Rand**. Algoritmus **AlgRefAtom9Sort** má od Algoritmu **AlgRefAtom9Rand** vzestupně seřazené indexy elementů v jednotlivých imaginárních barevných třídách v poli *items*.

Zobrazme nyní Tabulku 22.

Algoritmus	Čas sestavení matic $\mathbb{V}$ a $\mathbb{K}$	Absolutní a relativní chyba	Parametry
Model koule pokrytý síti o 10242 uzlech, 20480 elementech a 30720 hranách			
AlgReferenceAtomic	$\mathbb{V}$: 11.65 s, $\mathbb{K}$: 13.21 s	Abs: 5.80e-05, Rel: 3.90e-06	auto
AlgColoringH	$\mathbb{V}$: 12.24 s, $\mathbb{K}$: 13.18 s	Abs: 5.80e-05, Rel: 3.90e-06	auto
AlgRefAtomVec1	$\mathbb{V}$: 11.76 s, $\mathbb{K}$: 13.34 s	Abs: 5.80e-05, Rel: 3.90e-06	auto
AlgRefAtomVec1Rand	$\mathbb{V}$: 13.18 s, $\mathbb{K}$: 13.16 s	Abs: 5.80e-05, Rel: 3.90e-06	auto
AlgRefAtom9Rand	$\mathbb{V}$: 14.47 s, $\mathbb{K}$: 13.19 s	Abs: 5.80e-05, Rel: 3.90e-06	auto
AlgRefAtom9Sort	$\mathbb{V}$: 13.68 s, $\mathbb{K}$: 13.16 s	Abs: 5.80e-05, Rel: 3.90e-06	auto
AlgColoringHImprove	$\mathbb{V}$: 12.20 s, $\mathbb{K}$: 13.14 s	Abs: 5.79e-05, Rel: 3.90e-06	auto
Model koule pokrytý síti o 40962 uzlech, 81920 elementech a 122880 hranách			
AlgReferenceAtomic	$\mathbb{V}$: 190.43 s, $\mathbb{K}$: 213.01 s	Abs: 5.79e-06, Rel: 3.90e-07	auto
AlgColoringH	$\mathbb{V}$: 196.35 s, $\mathbb{K}$: 213.25 s	Abs: 5.79e-06, Rel: 3.90e-07	auto
AlgRefAtomVec1	$\mathbb{V}$: 191.43 s, $\mathbb{K}$: 212.42 s	Abs: 5.79e-06, Rel: 3.90e-07	auto
AlgRefAtomVec1Rand	$\mathbb{V}$: 235.36 s, $\mathbb{K}$: 213.09 s	Abs: 5.79e-06, Rel: 3.90e-07	auto
AlgRefAtom9Rand	$\mathbb{V}$: 238.07 s, $\mathbb{K}$: 212.70 s	Abs: 5.79e-06, Rel: 3.90e-07	auto
AlgRefAtom9Sort	$\mathbb{V}$: 227.36 s, $\mathbb{K}$: 212.47 s	Abs: 5.79e-06, Rel: 3.90e-07	auto
AlgColoringHImprove	$\mathbb{V}$: 195.77 s, $\mathbb{K}$: 213.25 s	Abs: 5.79e-06, Rel: 3.90e-07	auto
Model krychle pokrytý síti o 12290 uzlech, 24576 elementech a 36864 hranách			
AlgReferenceAtomic	$\mathbb{V}$: 16.91 s, $\mathbb{K}$: 19.25 s	Abs: 1.96e-04, Rel: 1.32e-05	auto
AlgColoringH	$\mathbb{V}$: 17.62 s, $\mathbb{K}$: 19.22 s	Abs: 1.96e-04, Rel: 1.32e-05	auto
AlgRefAtomVec1	$\mathbb{V}$: 17.10 s, $\mathbb{K}$: 19.42 s	Abs: 1.96e-04, Rel: 1.32e-05	auto
AlgRefAtomVec1Rand	$\mathbb{V}$: 19.53 s, $\mathbb{K}$: 19.34 s	Abs: 1.96e-04, Rel: 1.32e-05	auto
AlgRefAtom9Rand	$\mathbb{V}$: 20.97 s, $\mathbb{K}$: 19.17 s	Abs: 1.96e-04, Rel: 1.32e-05	auto
AlgRefAtom9Sort	$\mathbb{V}$: 19.97 s, $\mathbb{K}$: 19.33 s	Abs: 1.96e-04, Rel: 1.32e-05	auto
AlgColoringHImprove	$\mathbb{V}$: 17.62 s, $\mathbb{K}$: 19.25 s	Abs: 1.96e-04, Rel: 1.32e-05	auto
Model krychle pokrytý síti o 49154 uzlech, 98304 elementech a 147456 hranách			
AlgReferenceAtomic	$\mathbb{V}$: 275.13 s, $\mathbb{K}$: 306.45 s	Abs: 1.25e-04, Rel: 8.44e-06	auto
AlgColoringH	$\mathbb{V}$: 282.98 s, $\mathbb{K}$: 306.23 s	Abs: 1.25e-04, Rel: 8.44e-06	auto
AlgRefAtomVec1	$\mathbb{V}$: 275.69 s, $\mathbb{K}$: 307.45 s	Abs: 1.25e-04, Rel: 8.44e-06	auto
AlgRefAtomVec1Rand	$\mathbb{V}$: 340.61 s, $\mathbb{K}$: 307.31 s	Abs: 1.25e-04, Rel: 8.44e-06	auto
AlgRefAtom9Rand	$\mathbb{V}$: 346.61 s, $\mathbb{K}$: 307.71 s	Abs: 1.25e-04, Rel: 8.44e-06	auto
AlgRefAtom9Sort	$\mathbb{V}$: 328.79 s, $\mathbb{K}$: 307.92 s	Abs: 1.25e-04, Rel: 8.44e-06	auto
AlgColoringHImprove	$\mathbb{V}$: 281.16 s, $\mathbb{K}$: 306.23 s	Abs: 1.25e-04, Rel: 8.44e-06	auto

Tabulka 22: Testování času sestavení globálních matic $\mathbb{V}$ a $\mathbb{K}$ v knihovně BEM4I [11]

Z výsledků, které se nachází v Tabulce 22, vidíme , že při volbě parametru *auto* v Algoritmu **AlgReferenceAtomic** se nám podařilo dosáhnout o pár jednotek procent lepšího času, než při volbě parametru *dynamic, 8* z Tabulky 21. Dále vidíme, že při přístupu k indexům elementů

v poli *items* je vidět malé časové zhoršení, které není nijak velké, jedná se řadově o pár jednotek procent, jedná se o Algoritmus **AlgRefAtomVec1**. Velký vliv na běh výpočtu matice $\mathbb{V}$ má samotné pořadí vykonávaní elementů v jakém jsou vyhodnocovány. Jedná se o časové zhoršení kolem patnácti procent Algoritmu **AlgRefAtomVec1Rand** oproti Algoritmu **AlgReference-Atomic**. Mírné zhoršení času v jednotkách procent má i čekaní před pracováním nové imaginární *color* a *shade* barvu. To co naopak dokázalo snížit o pár jednotek procent výpočetního času sestavení matice $\mathbb{V}$ bylo seřízení indexů elementů v poli **items** v Algoritmu **AlgRefAtom9Sort** oproti Algoritmu **AlgRefAtom9Rand**.

Z výsledku v této kapitole nám vychází několik závěrů pro zlepšení času sestavení matice $\mathbb{V}$ pomocí barvícího Algoritmu **AlgColoringH**. Důležitou roli během vykonávání zde hraje pořadí zpracovaných elementů sítě. Svou zásluhu na tom má *cache* paměť procesoru a tzv. *cache line*. *Cache line* je souvislý, ale omezený úsek paměti, který procesor načte z paměti *RAM* do této *cache* paměti procesoru v domnění, že v blízké době bude přistupovat k prvkům uložených v *cache line*. Protože obarvené elementy stejné barvy nejsou zpravidla v paměti uložené za sebou, vzniká zde situace nazývána *cache miss* Jedná se o situaci, kdy budoucí zpracovávané elementy se nenachází v *cache line* a je nutné získat data o elementech sítě z *RAM* paměti a načíst nové data do *cache* paměti procesoru. Tato operace získání dat z *RAM* je časově náročnější, než prohledání dat uložených v *cache line*. Jednou z možností, jak lehce snížit čas sestavení matice $\mathbb{V}$ v Algoritmu **AlgColoringH** je vytvořit si pomocné pole *coordinate*. V tomto poli *coordinate* by si algoritmus uchovával informace o indexech uzlů elementů v pořadí, v jakém jsou uloženy indexy elementů v barevných třídách grafu T. Tím zvýšit efektivitu využití *cache line*. Případně další možností je setřídit indexy elementů v barevných třídách v Algoritmu **AlgColoringH**. Tento test, kdy setřídíme vzestupně indexy v barevných třídách Algoritmu **AlgColoringH** budeme v Tabulce 22 značit **AlgColoringHImprove**. Ideální by bylo, přeindexovat paměť na základě zpracování pořadí elementů, které vychází z barvení grafu T. Jedná se o velký zásah do knihovny BEM4I [11], který může být námětem v pokračování budoucí práce na podobné téma. Nevýhodu použití metody barvení při sestavování matice $\mathbb{V}$ je čekání vláken, před tím než se dokončí dřívější zpracovávání elementů *color* a *shade* barvy. Pro připomenutí pokud se Algoritmu **AlgColoringH** podaří dobře vrcholově obarvit graf T pomocí 9 barev, tak jako v testech v kapitole 7.1.1. Znamená to, že umíme dobře vrcholově obarvit graf H pomocí $9 \cdot 9 = 81$ barev. Proto v Algoritmu **AlgColoringH** vznikne 81 čekání. Počet čekání je stejný jako počet barev v dobrém vrcholovém barvení grafu H. Počet čekání, tj. barev v dobrém vrcholovém barvení grafu H, se dá možná zmenšit lepší heuristikou barvení grafu T. Dolní odhad klikového čísla grafu H je 36, což značí, že je třeba určitě použit 36 čekání. Pokud budeme chtít využít metodu dobrého vrcholového barvení při řešení kolizí během sestavení matice $\mathbb{V}$, čekání se nevyhneme.

7.3 Numerické experimenty barvení grafu T

V následující sekci si jednotlivé testovací Algoritmy pojmenujeme. Základní výpočetní kostrou bude Algoritmus 12 ze strany 77. Zaveďme si následující označení pro testované algoritmy.

- **AlgNaiveBoolean** budeme značit algoritmus, který bude sestaven z Algoritmu 12 ze strany 77. Tento algoritmus bude opatřen Algoritmem 16 ze strany 90 pro redukci barvy v barevných třídách a vyvažovacím Algoritmem 15 ze strany 88. Důležitou poznámkou je, že tento algoritmus ve svém barvení používá *boolean* pole v proměnné *usedColors* v Algoritmu 12.
- **AlgNaiveBitMasking** budeme značit algoritmus, který opět bude sestaven z Algoritmu 12 ze strany 77. Tento algoritmus bude také opatřen Algoritmem 16 ze strany 90 pro redukci barvy v barevných třídách a vyvažovacím Algoritmem 15 ze strany 88. Rozdílem oproti algoritmu označeným **AlgNaiveBoolean** je ten, že ve svém barvení v Algoritmu 12 bude využívat bitové operace ve struktuře *BitMasking* v proměnné *usedColors* z Výpisu 1 na straně 78.
- **AlgNeighborsBitMasking** budeme značit algoritmus, který opět bude sestaven z Algoritmu 12 ze strany 77. Tento algoritmus bude taktéž opatřen Algoritmem 16 ze strany 90 pro redukci barvy v barevných třídách a vyvažovacím Algoritmem 15 ze strany 88. Tento algoritmus bude využívat také bitové operace v proměnné *usedColors* v Algoritmu 12. Navíc v samotném barvení grafu T využijeme Algoritmus 13 ze strany 81. V Algoritmu 13 se následující barvící vrcholy v grafu T určují na základě sousedních vrcholů grafu T, které již byly zpracovány.
- **AlgClique4BitMasking** budeme značit algoritmus, který opět bude sestaven z Algoritmu 12 ze strany 77. Tento algoritmus bude taktéž opatřen Algoritmem 16 ze strany 90 pro redukci barvy v barevných třídách a vyvažovacím Algoritmem 15 ze strany 88. Tento algoritmus bude využívat také bitové operace v proměnné *usedColors* v Algoritmu 12. Navíc v samotném barvení grafu T využijeme Algoritmus 14 ze strany 82. V Algoritmu 14 se heuristicky barví kliky grafu T o velikosti 4.

Jako testovací model nám poslouží model koule, která bude pokrytá různě velkou sítí. Pod pojmenovaným sloupcem *Redukovaný počet barev* rozumíme počet barev v dobrém vrcholovém barvení grafu T, na který se podařilo heuristickému Algoritmu 16 snížit počet použitých barev. Dalším sloupcem *Počet použitých barev grafu T* rozumíme konečný počet použitých barev v dobrém vrcholovém barvení grafu T. Posledním sloupcem *Celkový čas algoritmu* rozumíme celkový čas od zpracování sítě až po samotné dobré vrcholové barvení grafu T. Samotné výsledky modelu koule jsou zobrazeny v Tabulce 23. V druhé Tabulce 24 jsou zobrazeny časy nejdůležitějších sekcí prováděných v algoritmech. Pod pojmem *Setavení grafu T* rozumíme sestavením polí *Node-*

NeighborsElementsIndex a *ElementNeighborsElementsIndex*. Sestavený graf T je reprezentován pomocí pole *ElementNeighborsElementsIndex*, které v sobě obsahuje pro každý vrchol grafu T množinu sousedních vrcholů. Dalším sloupcem v pořadí je *Barvení grafu T* Pod tímto pojmem rozumíme pouze čas samotného algoritmu, který dobře vrcholově obarví graf T. Pod pojmem *Redukce barev* se skrývá čas heuristického Algoritmu 16 ze strany 16. Pod posledním pojmem *Vyrovnání počtu vrcholů* rozumíme čas vykonávání Algoritmu 15 ze strany 88, který se pokusí o rovnoměrné vyvážení počtu vrcholů v barevných třídách.

Model koule pokrytý síti o 10242 uzlech, 20480 elementech a 30720 hranách			
Algoritmus	Redukovaný počet barev	Počet použitých barev grafu T	Celkový čas algoritmu
AlgNaiveBoolean	0	10	7 ms
AlgNaiveBitMasking	0	10	4 ms
AlgNeighborsBitMasking	0	9	9 ms
AlgClique4BitMasking	0	9	9 ms
Model koule pokrytý síti o 40962 uzlech, 81920 elementech a 122880 hranách			
AlgNaiveBoolean	0	10	32 ms
AlgNaiveBitMasking	0	10	31 ms
AlgNeighborsBitMasking	1	9	34 ms
AlgClique4BitMasking	0	9	42 ms
Model koule pokrytý síti o 163842 uzlech, 327680 elementech a 491520 hranách			
AlgNaiveBoolean	0	10	155 ms
AlgNaiveBitMasking	0	10	140 ms
AlgNeighborsBitMasking	0	9	162 ms
AlgClique4BitMasking	0	9	201 ms
Model koule pokrytý síti o 655362 uzlech, 1310720 elementech a 1966080 hranách			
AlgNaiveBoolean	0	10	750 ms
AlgNaiveBitMasking	0	10	669 ms
AlgNeighborsBitMasking	0	9	775 ms
AlgClique4BitMasking	0	9	927 ms

Tabulka 23: Model koule o různých velikostech sítí

Model koule pokrytý síti o 10242 uzlech, 20480 oblastech a 30720 hranách				
Algoritmus	Sestavení grafu T	Barvení grafu T	Redukce barev	Vyrovnání počtu vrcholů
AlgNaiveBoolean	5 ms	1 ms	0 ms	0 ms
AlgNaiveBitMasking	5 ms	0 ms	0 ms	0 ms
AlgNeighborsBitMasking	5 ms	1 ms	0 ms	0 ms
AlgClique4BitMasking	5 ms	1 ms	0 ms	0 ms
Model koule pokrytý síti o 40962 uzlech, 81920 oblastech a 122880 hranách				
AlgNaiveBoolean	25 ms	4 ms	1 ms	1 ms
AlgNaiveBitMasking	25 ms	1 ms	1 ms	1 ms
AlgNeighborsBitMasking	26 ms	4 ms	1 ms	1 ms
AlgClique4BitMasking	25 ms	4 ms	1 ms	1 ms
Model koule pokrytý síti o 163842 uzlech, 327680 oblastech a 491520 hranách				
AlgNaiveBoolean	117 ms	18 ms	5 ms	11 ms
AlgNaiveBitMasking	116 ms	9 ms	5 ms	11 ms
AlgNeighborsBitMasking	117 ms	29 ms	6 ms	12 ms
AlgClique4BitMasking	117 ms	31 ms	7 ms	11 ms
Model koule pokrytý síti o 655362 uzlech, 1310720 oblastech a 1966080 hranách				
AlgNaiveBoolean	527 ms	73 ms	49 ms	59 ms
AlgNaiveBitMasking	528 ms	33 ms	46 ms	59 ms
AlgNeighborsBitMasking	528 ms	126 ms	54 ms	62 ms
AlgClique4BitMasking	528 ms	139 ms	60 ms	61 ms

Tabulka 24: Model koule o různých velikostech sítí

7.3.1 Zhodnocení dobrého vrcholového barvení grafu T

Připomeneme, že všechny námi vytvořené algoritmy *AlgNaiveBoolean*, *AlgNaiveBitMasking*, *AlgNeighborsBitMasking* a *AlgClique4BitMasking* jsou jedno-vláknové algoritmy, které nevyužívají paralelismus. Pokud se podíváme na Tabulku 23, můžeme vidět, že časy algoritmů jsou dle původního plánu malé, vzhledem k velikosti počtu elementů sítě. V Tabulce 24 ve sloupci *Barvení grafu T* v algoritmech *AlgNaiveBoolean* a *AlgNaiveBitMasking* je více než poloviční rozdíl v času barvení. To je způsobeno využitím bitových operací popsaných v sekci 6.1 na straně 6.1 a využitých v Algoritmu *AlgNaiveBitMasking*. Dále v Tabulce 23 vidíme, že redukční Algoritmus 16 ze strany 90 je pro použití v Algoritmu *AlgClique4BitMasking* zbytečný, protože v Algoritmu 14 ze strany 82 se vždy podařilo najít v dobrému vrcholovém barvení grafu T 9 barev. Výpočetně nejnáročnější částí, která zabrala nejvíce výpočetního času z Tabulky 23, je sestavení samotného grafu T. Nejrychlejším algoritmem v našem testu je *AlgNaiveBitMasking*. Nevýhodou tohoto

Algoritmu *AlgNaiveBitMasking* je horší počet nalezených barev v dobrém vrcholovém barvení grafu T. Oproti tomu algoritmům *AlgNeighborsBitMasking* a *AlgNeighborsBitMasking* se vždy podařilo nalézt 9 barev v dobrém vrcholovém barvení grafu T. Pokud tedy z Algoritmu *AlgClique4BitMasking* odebereme redukční Algoritmus 16 dostáváme téměř shodný čas s Algoritmem *AlgNeighboursBitMasking*. Proto jsme tento Algoritmus *AlgClique4BitMasking* upravili na Algoritmus *AlgColoringH* a využili v předchozí sekci 7.3.

7.4 Shrnutí

Při použití barevných tříd grafu H se nám podařilo dosáhnout podobných časů sestavení matice $\mathbb{V}$ jako v referenčním řešením *AlgReferenceAtomic* při výpočtu Laplaceovy úlohy v metodě hraničních prvků. Námi navržené řešení je časově pomalejší řádově v jednotkách procent. Do 20000 elementů sítě je naše řešení časově shodné s referenčním řešením při sestavení matice $\mathbb{V}$. Svůj význam a použití námi navržená metoda barvení grafu H může nalézt v paralelním výpočtu Laplaceovy úlohy metodou hraničních prvků na grafických procesorech (GPU). Důvod je ten, že v současné době lze se realizovat ošetření kolizí pomocí atomické operace na grafických procesorech, ale synchronizace se týká pouze vláken ve stejném bloku bohužel ne však všech vláken, které jsou aktivní [3]. Proto je ponechána tato část na programátorovi, aby ošetřil vznik kolizí napříč všemi vlákny. Navíc je tento způsob limitován velikostí na operaci 64 bitových slov. Při větší přesnosti musí programátor sám zajist ošetření kolizí. Řešení je použít námi navržené barvení v GPU architekturách. Což je i jeden z dalších možných směrů, kam se budoucí práce na toto téma mohou vydat. I přes to, že se nám nepodařilo zrychlit současné sestavení matice $\mathbb{V}$ v metodě hraničních prvků, ale za to se nám podařilo vyrovnat implementaci, která využívá hardwarové podpory tzv. atomické sekce. Dále se nám podařilo navrhnout efektivní algoritmus pro dobré vrcholové barvení grafu H. Časová a prostorová náročnost sestavení barevných tříd grafu H našim Algoritmem **AlgColoringH** je $\mathcal{O}(\sqrt{|N_H|}) = \mathcal{O}(F)$, kde F je počet elementů sítě. V testech, kde sestrojené grafy H vznikly ze sítí, ve kterých každý uzel sítě byl součásti právě šesti elementů, je stupeň každého vrcholu v grafu H 168. Dolní odhad počtu barev v dobrém vrcholovém barvení grafu H je 36 na základě Věty 1 o klikovém číslu grafu. Horní odhad počtu barev v dobrém vrcholovém barvení je 168 podle Brooksovy věty 3. Nám se v testech podařilo nalézt obarvení grafu H pomocí $9 \cdot 9 = 81$ barev proto, že graf T se podařilo dobře vrcholově obarvit pomocí 9 barev. Čím méně barev bude použito na dobré vrcholové barvení, tím větší podúlohy budou moci být vykonávány paralelně, neboť vlákna budou moci zpracovat více úloh bez zbytečného čekání. Počet barev použitých na dobré vrcholové barvení grafu H má vliv na počet čekání mezi vlákny. Některá rychlá vlákna čekají před zpracováním nové barvy na vlákna, která ještě zpracovávají úlohy předešlé barvy grafu H. Možná lze eliminovat počet barev grafu T lepším barvícím heuristickým algoritmem, avšak chceme stále mít zaručeno, že obarvení grafu T bude rychlé.

8 Závěr

Práce je zaměřená na efektivní implementaci paralelní metody hraničních prvků v knihovně BEM4I [11] bez použití tzv. atomické operace. V této práci jsme pomocí teorie grafů v kapitole 4 popsali problém s kolizemi. Tento problém může vzniknout, když dvě lokální matice se snaží paralelně zapsat svá data do globální matice $\mathbb{V}$ a během zápisu přistupují ke společné buňce v matici $\mathbb{V}$. Za modelovou úlohu jsme v této práci použili Laplaceovou úlohu, kterou řešíme pomocí metody hraničních prvků. Za bázové funkce jsme uvažovali po částech lineární funkce. Naším cílem je navrhnout efektivní algoritmus, který by rozdělil množinu dvojic elementů, na kterých interagují bázové funkce, do disjunktních množin. Podmínkou, kterou klademe na každou takovou množinu, je, že libovolná interakce dvou dvojic elementů ze stejné množiny nezapisuje výsledky ze své lokální matice do stejných buněk v globální matici $\mathbb{V}$. Díky této podmínce mohou být lokální matice počítaný paralelně na dvojicích elementů ze stejné množiny, bez toho aby vznikaly kolize.

V kapitole 4 se věnujeme návrhu algoritmu, který by rozdělil dvojice elementů do nezávislých množin tak, aby paralelní výpočty mohly běžet pro elementy ze stejné množiny bez vzniku kolizí. Hlavní myšlenka je založená na dobrém vrcholovém barvení grafu H, kde elementy obarvené stejnou barvou mohou být vykonávány paralelně. Graf H popisuje vztahy mezi dvojicemi elementů, pro které výpočet nemůže být proveden paralelně, neboť by došlo ke kolizi. Je potřeba si uvědomit, že počet vrcholů grafu H odpovídá uspořádaným dvojicím elementů sítě, kterých je F^2, kde F značíme počet elementů sítě. Pro představu, máme-li 3D objekt pokrytý sítí, kde každý uzel je součástí právě šesti elementů (trojúhelníků), pak každý vrchol grafu H má stupeň vrcholů 168, tj. každý zápis lokální matice do matice $\mathbb{V}$ je v kolizi s dalšími 168 zápisy jiných matic. Nám se v této práci podařilo vymyslet a úspěšně implementovat vlastní Algoritmus 12, který určí barevné třídy grafu H s časovou a paměťovou náročností $\mathcal{O}(F)$. Algoritmus 12 využívá triku, kdy se barví menší graf T. Vrcholy grafu T odpovídají elementům sítě a hrana mezi elementy existuje v případě, že elementy sdílí společný vrchol. Samotná funkčnost Algoritmu 12, že opravdu řeší problém s kolizemi, je podložena Větou 7 autora této práce a nachází se na straně 65.

V kapitole 6 jsme se zabývali vylepšeními Algoritmu 12. Jedná se například o využití bitové reprezentace čísel při častém zjišťování informace o nejnižší nepřiřazené barvě vrcholu na svém okolí. Dále v této kapitole 6 zmiňujeme vlastní heuristický algoritmus, který slouží k vyvážení počtu vrcholů v jednotlivých barevných třídách grafu T. S tím je spojené i vyvážení počtu vrcholů v jednotlivých barevných třídách grafu H. To má vliv na efektivní použití paralelismu během sestavení matice $\mathbb{V}$. Díky tomu jsou velikosti úloh napříč barvami grafu H rozdělené rovnoměrné a každé vlákno zpracuje stejně velikou podúlohu. Minimalizuje se díky tomu čas čekání před zpracováním elementů nové barvy. Velká část kapitoly 6 je zaměřena na popis vlastních

heuristických barvících algoritmů, které se snaží snížit počet barev v dobrém vrcholovém barvení grafu T. Počet použitých barev na dobré obarvení grafu T má kvadratický vliv na počet použitých barev v dobrém vrcholovém barvení grafu H. Proto se vyplatí vynaložit úsilí při barvení menšího grafu T.

Kapitola 7 je věnována numerickým experimentům, které jsme prováděli za pomocí knihovny BEM4I [11]. Jedním z cílů této práce byla samotná implementace vlastních algoritmů do této knihovny, což se nám podařilo. Všechny testy zmíněné v kapitole 7 byly prováděny na jednom výpočetním uzlu superpočítače Barbora [11], kde běželo 36 vláken. V první části kapitoly 7 porovnávám časy sestavení matice $\mathbb{V}$ naší implementací řešení kolizí s již implementovaným řešením, které využívá ošetření problémového místa v kódu pomocí atomické operace. Testy v této sekci řešily Laplaceovou úlohu ve 3D metodou hraničních prvků. Opět i zde jsme využili po částech lineární bázové funkce. Pokud síť objektu obsahovala 20000 elementů, obě metody měly stejný čas sestavení matice $\mathbb{V}$. Při větším počtu elementů sítě bylo rychlejší stávající řešení kolizí pomocí atomické sekce o pár jednotek procent. V druhé sekci této kapitoly 7 jsme se zabývali příčinami vzniku zpoždění sestavení matice $\mathbb{V}$ při použití metody barvení. Zjistili jsme, že velký dopad na čas sestavení matice $\mathbb{V}$ má uspořádání elementů v paměti. Pokud by elementy v paměti byly přeuspořádány v závislosti na pořadí elementů v barevných třídách, mohlo by být sestavení matice $\mathbb{V}$ rychlejší pomocí metody barvení. V třetí části této kapitoly 7 jsme se zabývali časy obarvení různých sítí a testovali jsme různá vylepšení Algoritmu 12. Například model koule, který byl složen 10^6 elementů, zvládl vylepšený Algoritmus 12 rozdělit 10^{12} uspořádaných dvojic elementů do 81 barevných tříd grafu H za 927 ms.

Námi navržené a implementované řešení kolizí nebylo rychlejší, než dosavadní řešení, které využívá atomickou operaci v knihovně BEM4I [11], avšak doba výpočtů se lišila nepatrně v jednotkách procent. Svůj význam tento grafový přístup nalezne při řešení stejné úlohy na GPU architektuře, kde sice existuje podpora atomické operace v rámci jednoho bloku vláken (thread bloku), ale ne však synchronizace napříč všemi vlákny jiných blocích [3]. Algoritmus 12 lze snadno modifikovat i pro použití jiného typu bázových funkcí než jsou po částech lineární. Taktéž lze Algoritmus 12 upravit a použít pro jiný tvar elementů než jsou trojúhelníky.

Literatura

1. BOUCHALA, Jiří. *ÚVOD DO FUNKCIONÁLNÍ ANALÝZY* [online]. 2012 [cit. 2020-04-01]. Dostupné z: `http://mi21.vsb.cz/sites/mi21.vsb.cz/files/unit/uvod_do_funkcionalni_analyzy.pdf`.
2. BOUCHALA, Jiří. *VARIAČNÍ METODY* [online]. 2012 [cit. 2020-04-01]. Dostupné z: `http://mi21.vsb.cz/sites/mi21.vsb.cz/files/unit/variacni_metody.pdf`.
3. *CUDA C++ PROGRAMMING GUIDE* [online]. 2019 [cit. 2020-04-01]. Dostupné z: `https://docs.nvidia.com/pdf/CUDA_C_Programming_Guide.pdf`.
4. *Git* [online]. 2020-02-23 [cit. 2020-04-01]. Dostupné z: `https://git-scm.com/`.
5. HAMMACK, Richard; IMRICH, Wilfried; KLAVŽAR, Sandi. *Handbook of Product Graphs*. CRC Press, 2011. Discrete Mathematics and Its Applications. ISBN 9781439813058.
6. *IT4Innovations: Documentation Introduction* [online] [cit. 2020-04-01]. Dostupné z: `https://docs.it4i.cz/introduction/`.
7. KOVÁŘ, Petr. *Teorie grafů, učební text* [online]. 2020 [cit. 2020-02-23]. Dostupné z: `http://homel.vsb.cz/~kov16/files/skriptum_teorie_grafu_rozsirene_tisk.pdf`.
8. KUBALE, Marek; ENGLISH, Optymalizacja Dyskretna; SOCIETY, American Mathematical. *Graph Colorings*. American Mathematical Society, 2004. Contemporary mathematics (American Mathematical Society) v. 352. ISBN 9780821834589.
9. LEWIS, Rhyd. *A Guide to Graph Colouring: Algorithms and Applications*. Springer International Publishing, 2015. ISBN 9783319257303.
10. MAREŠ, Martin; VALLA, Tomáš. *Průvodce labyrintem algoritmů*. CZ.NIC, 2017. ISBN 788088168225. Dostupné také z: `https://knihy.nic.cz/files/edice/pruvodce_labyrintem_algoritmu.pdf`.
11. MERTA, Michal; ZAPLETAL, Jan. *BEM4I: IT4Innovations* [online]. VŠB –Technical University of Ostrava, 17. listopadu 2172/15, 708 00 Ostrava-Poruba, Czech Republic [cit. 2020-04-01].
12. *OpenMP 5.0 API Syntax Reference Guide* [online]. 2019 [cit. 2020-04-01]. Dostupné z: `https://www.openmp.org/wp-content/uploads/OpenMPRef-5.0-0519-web.pdf`.
13. RJASANOW, Sergej; STEINACH, Olaf. *The Fast Solution of Boundary Integral Equations*. Springer US, 2007. Mathematical and Analytical Techniques with Applications to Engineering. ISBN 9780387340425.
14. SADOWSKÁ, Marie. *Úvod do (klasických) Boundary Element Methods pro 2D a 3D Laplaceovu rovnici* [online]. 2011 [cit. 2020-04-01]. Dostupné z: `https://homel.vsb.cz/~sad015/students/materials/uvod_BEM.pdf`.

15. WEST, Douglas Brent. *Introduction to Graph Theory*. Prentice Hall, 2001. Featured Titles for Graph Theory Series. ISBN 9780130144003.